PART FIVE

Indicator Examples

Part 5 of Practical Software and Systems Measurement provides examples of how measurement indicators can be applied to systematically analyze project issues.

This part of the Guide is organized into three chapters:

- **Chapter 1, Indicator Generation**, discusses ways of representing indicators.
- **Chapter 2, Single Indicator Examples**, presents examples of PSM indicators with explanations of how they were generated and analyzed for project insight.
- **Chapter 3, Integrated Indicator Examples**, presents examples of how complex or multiple indicators are used in a thorough analysis of a particular situation or scenario.

[This page intentionally left blank.]

1

Indicator Generation

An *indicator* is a measure or combination of measures that provide insight into an issue or concept. Most indicators compare actual values with baselines (plans or targets). Indicators can be displayed visually to facilitate analysis. When designing presentations of indicators, be sure that the data displays are clear and accurate. Two tasks are required to represent an indicator effectively: 1) identify the specific measurement data items to display; and 2) define the format for presentation of the data items.

Indicators may be represented in many ways, ranging from simple data tables to graphical representations. Graphical representation usually works best for comparing actual dates with baselines. Graphs show trends, variances, and relationships more clearly than do tables of numbers.

1.1 Graphical Representation Techniques

Many simple charting techniques produce graphical representations of measurement data. The ability to extract pertinent information from measurement data can be improved with proper selection and use of these charting techniques. The three most commonly used charting techniques are line, bar, and scatter charts, as described below:

Line Charts, sometimes called run charts, represent a series of measurement data values over time. A series may contain actual or expected values. Each value in the series is reported for a specific point in time. Values are plotted as points on the graph, and the values are connected with lines to show progress or a trend. For example, a line chart may include one series of planned values that shows the cumulative number of components scheduled to complete coding and unit testing each month during a six-month period. As components are completed, a second series of values is added to the graph each month, providing a comparison between measured and expected values. Figure 5-1 is an example of a line chart.

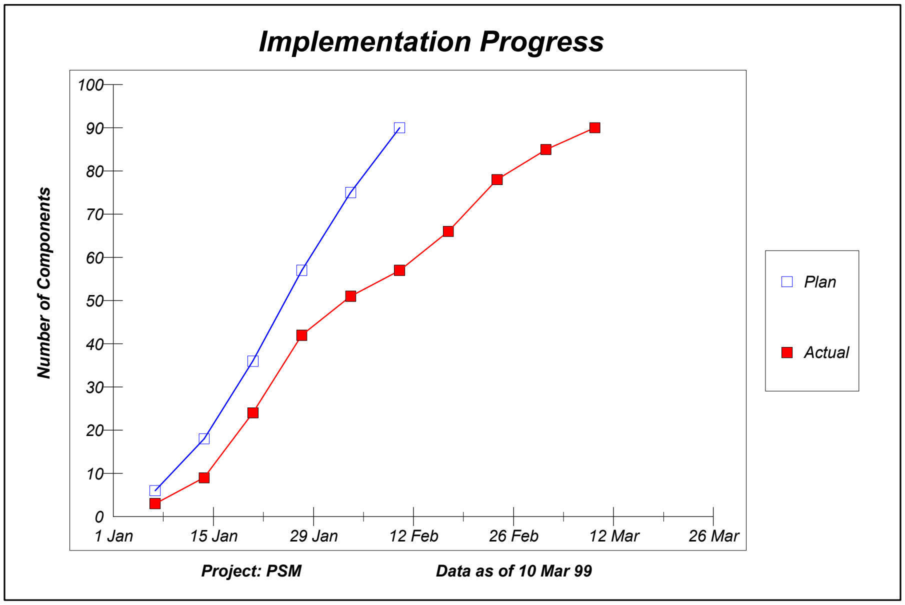

Figure 5-1. Sample Line Chart

Bar Charts represent the count or frequency of a set of components or events. Bars are typically drawn vertically, with the Y-axis indicating the number of components or events being counted. Each bar contains data associated with a class or grouping. For example, the bar might represent the number of defects detected for each product component or within each phase of the project life cycle. Sets of bars may compare two series, such as measured and expected values. Figure 5-2 is an example of a bar chart. A target, limit, or average may be superimposed as a straight horizontal line to represent the baseline.

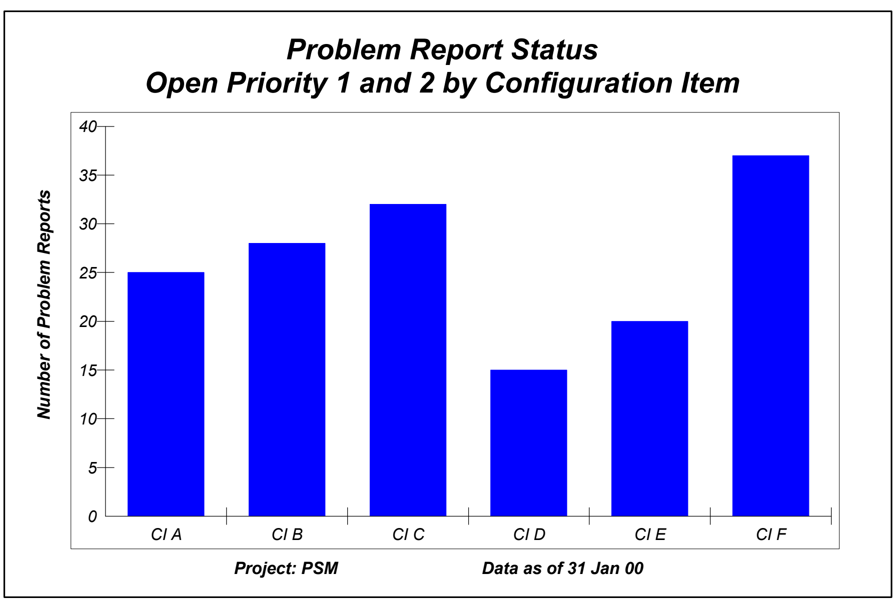

Figure 5-2. Sample Bar Chart

Scatter Charts graph possible relationships between two factors. They plot a series of points based on the values of the two factors. For example, each point may represent historical data from a completed project. To explore the size-schedule relationship, size might be assigned to the x-axis and schedule might be assigned to the y-axis. When all points are plotted, the strength of the relationship is illustrated by the tendency of the points to cluster along a line. Figure 5-3 is an example of a scatter chart. The relationship in this example is presented as a straight-line linear regression.

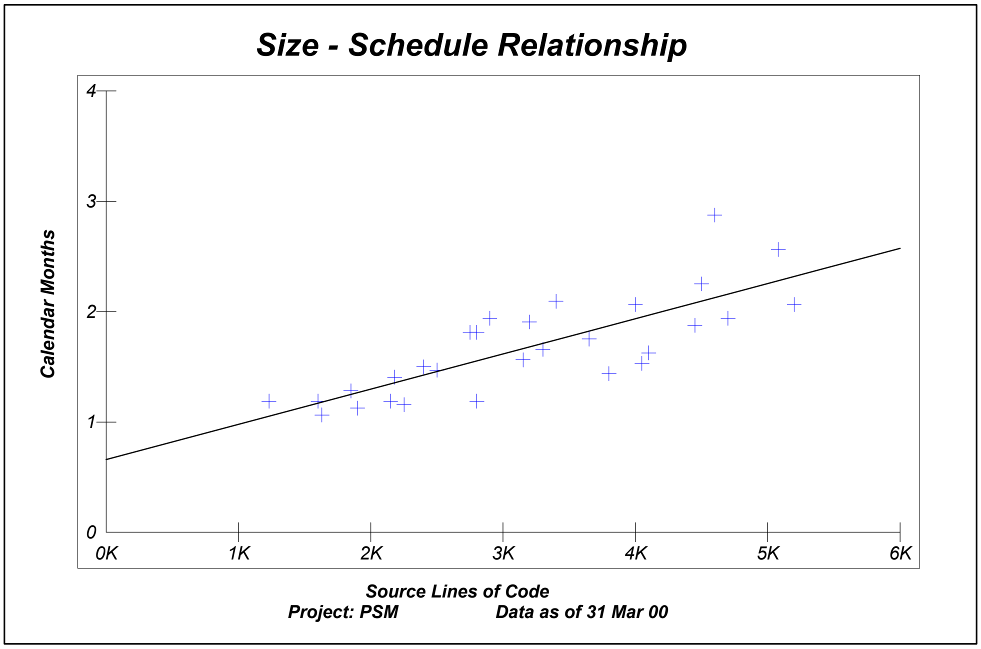

Figure 5-3. Sample Scatter Chart

Stoplight Charts may be used to summarize status in terms of many different measures simultaneously. Colors represent performance relative to plan. Colors must be assigned based on predefined criteria. Recognize that, as summaries, stoplight charts don't provide the trend information needed to diagnose problems and predict project outcomes.

In Figure 5-4, status is indicated by the color of the lights:

- A green light indicates subsystems that have been accepted by the customer.
- A yellow light indicates subsystems that have not yet been accepted and that have no anticipated problems, or that have possible problems with work-arounds.
- A red light indicates subsystems that have problems or a high probability of problems.

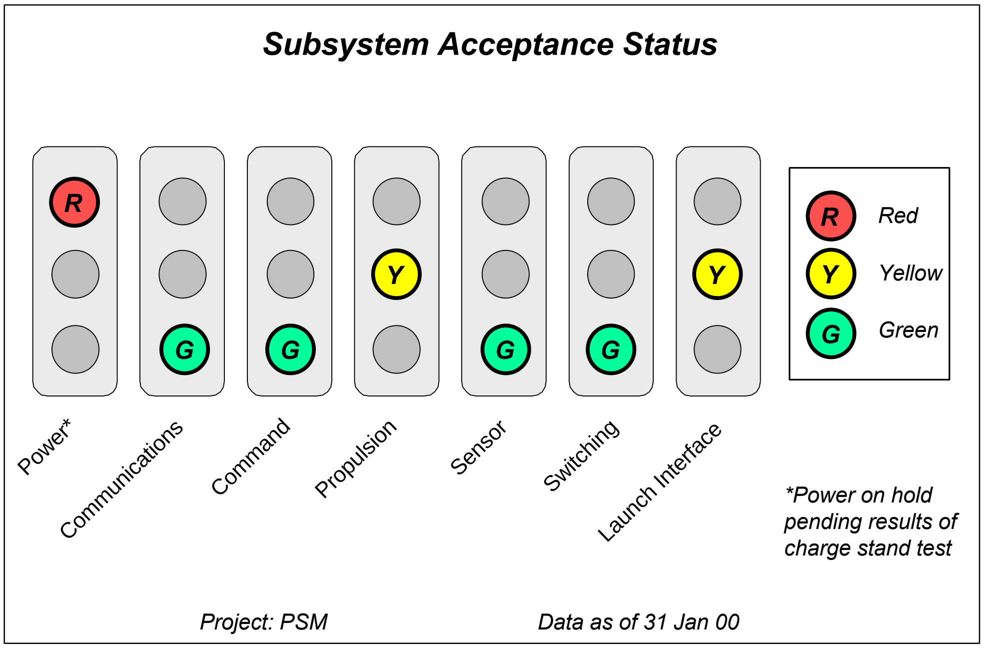

Figure 5-4. Subsystem Acceptance Status

1.2 Indicators as Estimators

An estimator is a special type of indicator that compares two different measures rather than two values of one measure. In this case, values of one measure are used to estimate or predict the values of the other measure. PSM estimators generally include three factors:

- The known or assumed value of a measure is sometimes referred to as the independent variable.
- The estimated or predicted value of another measure is sometimes referred to as the dependent variable.
- Uncertainty associated with the estimate is an indication of the degree of confidence in the relationship.

Figure 5-5 illustrates an estimator (Putnam, et al., 1992) that reviews the results of another estimating approach. In this example, product size (the horizontal axis) is used to estimate effort (the vertical axis). The trend line (solid diagonal) shows the relationship between size and effort. As size increases, effort also increases. There is a 95% confidence limit around the trend line. Ninety-five percent of the time, actual effort should fall within this range for any given size. The remaining uncertainty comes from two sources: 1) natural variability in the estimating relationship (resulting from variations in the performance of people and organizations) and 2) lack of information on the correct estimating relationship (possibly because data from similar past projects is not available).

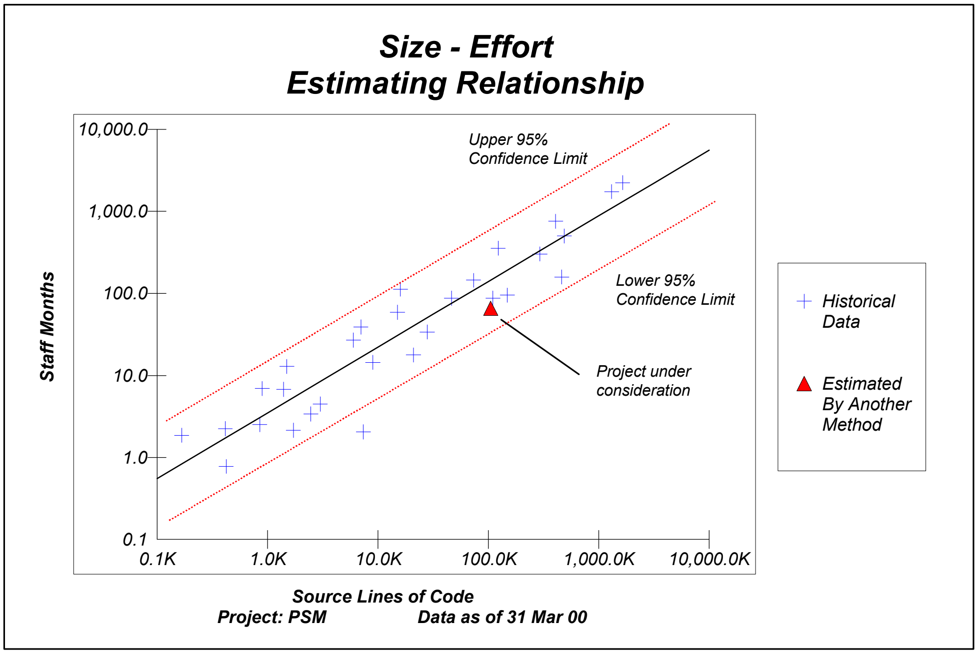

Figure 5-5. Example of an Estimator

While some measures are closely associated with specific indicators, the PSM concept of an indicator promotes the combination of measures in many different ways. Selecting a measure does not necessarily determine the indicators that will be produced from it, since indicators are designed to suit specific information needs.

1.3 Communicating Information on Graphs

Well-designed representations of indicators facilitate communication of measurement results. Graphs should convey a clear message and should not be too complex. It is better to have many graphs than to have several messages on one graph.

Guidelines for developing effective indicator graphs include:

- Provide a descriptive title to identify the indicator name, type of data, and component (if applicable) represented by the indicator.
- Include the project name or unique identifier.
- Be sure that axis labels include type of components and scale markers, such as dates or counts.
- For graphs showing time trends, mark major milestones or significant events occurring during the timeframe presented.
- Limit the amount of information shown on a single graph, so that contrasts and comparisons can be made easily.

- Place a “data as of” date on every graph to show the reporting period in which the data was collected. Many graphs show plans or projections beyond the current reporting period or the “as of” date. If the date that the graph was generated is important, include that date.

- Identify the source of the data, and include the version number of documents.

- Label significant items and trends in the data.

Guidelines on graph format include:

- To show trends, use the connect-the-dots technique rather than curve-fitting.

- Use contrasting styles for lines, bars, and data points to represent different data groups. Use something other than color or shading (such as texture or another fill content) so that charts printed in black and white can be easily understood.

- Test color graphs in all the intended media, such as different computer systems, projectors, and printers. Since different media render colors differently, distinctions may be less obvious in one medium than another, and adjustments may be needed.

- If possible, label lines, bars, and data points directly on the graph. Otherwise, use a key that associates a label with each contrasting style of line, bar, or data point.

- Use similar conventions for graphs within a report or project, such as solid boxes for actuals and open boxes for plans. Use the same colors or textures for similar data appearing on multiple charts.

- Define end points on axes to show expected ranges of plotted data.

- Check that the use of percentages does not hide significant trends in the data.

- When comparing two indicator graphs, use the same axes on both graphs.

- If showing multiple lines on the same graph, make sure that the scales are appropriate to distinguish significant features of each line. For example, if one line requires a smaller range than another line, it may be better to use two different graphs, rather than to combine the lines.

The graph in Figure 5-6 illustrates some of these guidelines.

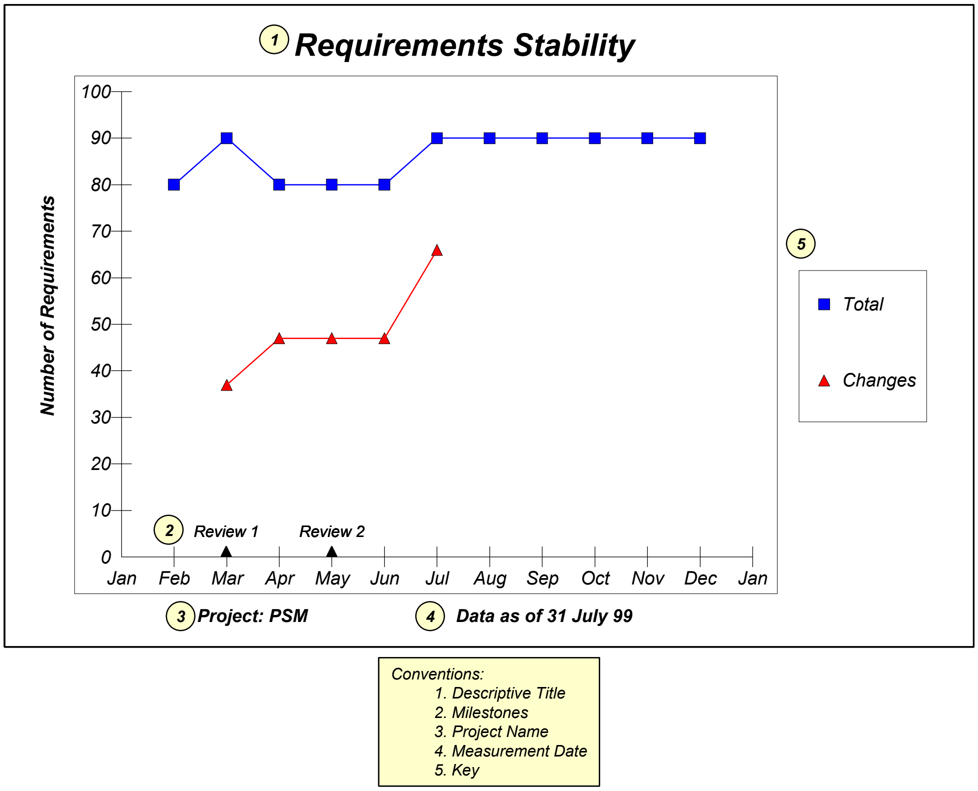

Figure 5-6. PSM Guidelines for Graph Formats

1.4 Example of a Graphical Representation of an Indicator

In this example, a banking system is being developed to provide a network of automated teller machines (ATMs). A critical design objective is that the ATMs must complete a user's transaction in less than 50 seconds (0.83 minutes). That was determined to be the maximum acceptable delay based on customer surveys.

A prototype system was completed and tested to measure system response times. Each of the tests recorded the transaction size, the transaction rate, and the system response time. The data component collected for transaction size was the number of components in data records that must be accessed to complete a transaction. Transaction rate was measured as the number of transactions that were concurrently processed by the network during one second. The system response time was the maximum time to complete a user's transaction. In these tests, transaction size and transaction rate were the independent attributes that were varied to measure the impact on the system response time. The results of these tests are recorded in Figure 5-7. System response time was measured in fractions of a minute and recorded in the matrix.

Response Time (sec) for Different Transaction Sizes and Transaction Rates										
	Transaction	Transaction Rate (Transactions /Sec)								Average Response Time
Test #	Size	15	20	25	30	35	40	45	50	
1	96	0.21	0.23		0.24			0.25		0.23
2	144	0.39		0.43	0.43		0.43		0.43	0.42
3	192		0.45			0.47				0.46
4	240	0.51		0.49			0.48		0.54	0.51
5	288		0.67		0.68	0.66				0.67
6	336						0.70	0.71		0.71
7	384	0.79						0.80		0.80
8	432		0.83		0.84	0.86			0.85	0.85
9	480	0.88				0.88	0.88		0.88	0.88

Figure 5-7. Measured Response Time for Transaction Processing Tests

This table of numbers suggests a problem, but a graph could show it more clearly. Figure 5-8 plots the average response time from each test against the target value. The latest results exceed the target. Attempting to understand the source of the problem led the system engineer to combine this data differently to perform additional analysis.

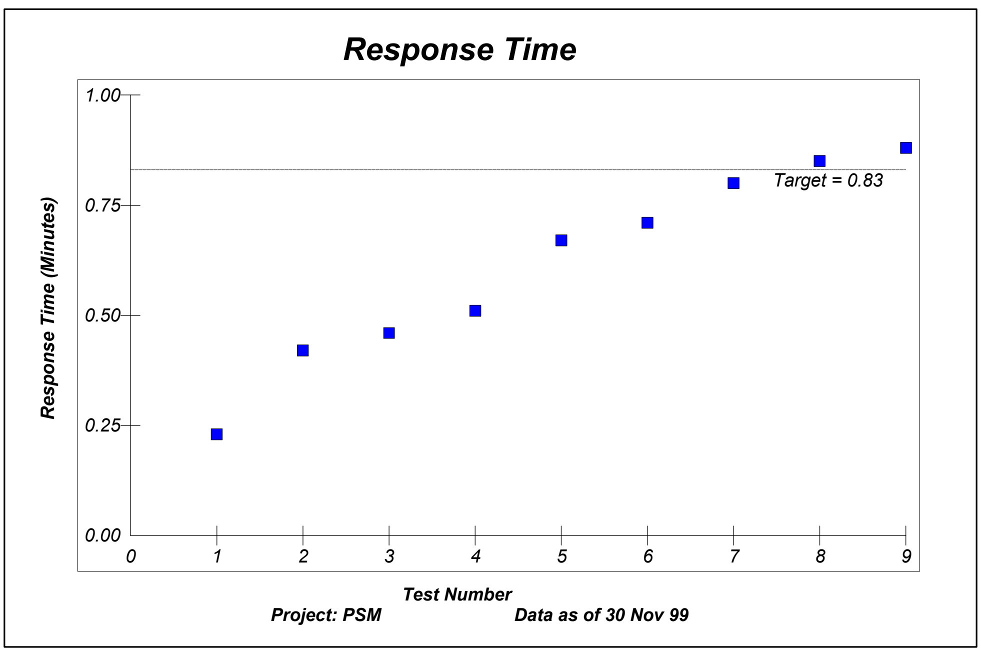

Figure 5-8. Response Time

It was initially assumed that the system response time was directly related to the number of transactions per second that the system must process. The more transactions arriving per second, the longer the average system response time.

Figure 5-9 shows the impact of transaction rate on system response time by graphing the transaction rate (the independent variable) against the system response time (the dependent variable). There does not seem to be a correlation between transaction rate and system response time because the points appear to be randomly distributed.

Figure 5-9. System Response Time versus Transaction Rate

However, Figure 5-10 shows system response time relative to transaction size. In this graph, there seems to be a relationship between these attributes. Figure 5-10 graphs the same data from Figure 5-7, but uses transaction size as the independent variable. This figure shows a strong, nearly linear correlation between these attributes. This example shows that the original assumption - the relationship between rate of transactions and the system response time - was wrong. Test data shows that transaction size has a strong influence on system response time, while transaction rate has a minimal influence. This helps the engineers adjust the design to minimize transaction size and response time.

This correlation can only be cited for the range of transaction sizes and rates that were used in this test. Larger transaction sizes may not show a linear trend, or significant increases in system response time may be observed above some transaction rates.

This example shows how the measures used to track the basic project issue, response time, could be combined in different ways to gain more detailed insight into an identified problem.

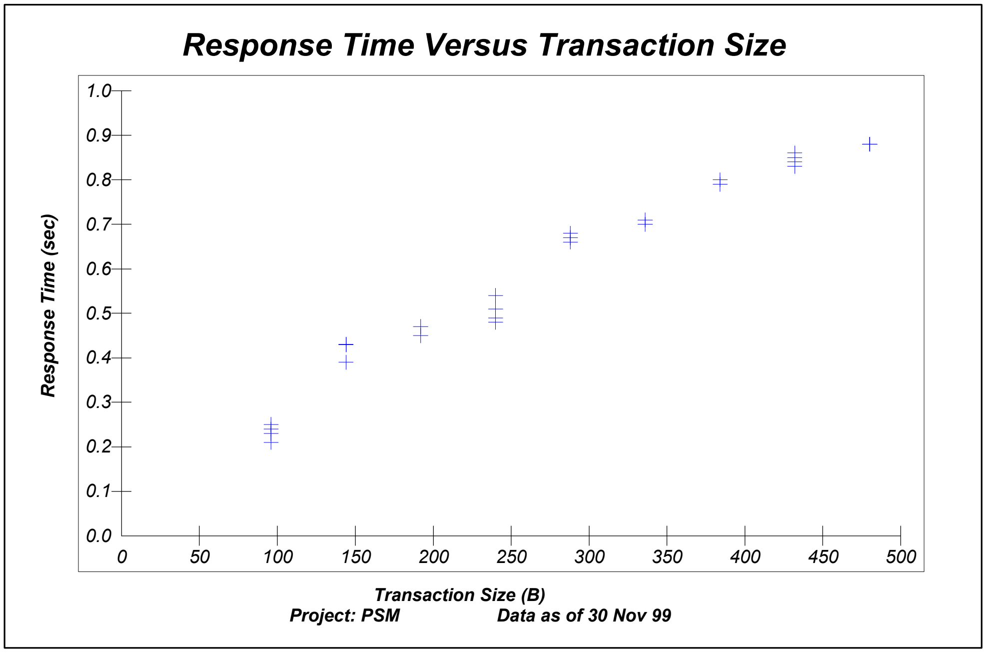

Figure 5-10. System Response Time versus Transaction Size

The following chapters in this part of the Guide contain indicator examples, typical graphs, and explanations of how to interpret them.

2

Single Indicator Examples

This chapter presents examples of PSM indicators with explanations of how they were generated, represented, and analyzed for project insight. At least one example is included for each PSM measurement category. Note that these are examples only; they do not represent a definitive set for all projects.

Each example indicator contains the following information:

- **Measure Name** - the title of each sub-section, reflecting the measurement data used for generating the indicator(s)
- **Category** - the PSM measurement category that best matches the information needed
- **Common Issue Area** - the PSM common issue area that is addressed in the analysis
- **Applicability** - whether it is appropriate for software and/or systems, or for most types of projects. Examples that apply to most types of projects include indicators for process improvement efforts.
- **Analysis Guidance and Examples** - information on how the indicator can be used during estimation, how the feasibility of plans can be assessed, and how project performance can be analyzed (where appropriate)
- **Additional Analysis** - other factors to consider when using this indicator
- **Lessons Learned** - industry experience in the use of this measure and/or indicator(s)

Table 5.11 indexes the indicator examples in this chapter. Each example indicator is identified as a numbered section and is cross-referenced to the PSM Common Issue Area, Measurement Category, and Measure.

Common Issue Area	Measurement Category	Measure	Section	Indicator(s)
Schedule and Progress	Milestone Performance	Milestone Dates	2.1	• Development Milestone Schedule • Milestone Progress - Maintenance Activities
	Work Unit Progress	Problem Report Status	2.2	• Problem Report Status • Problem Report Aging - Open Problem Reports • Problem Report Status - Open by Priority • Problem Report Status - Open Priority 1 and 2 by Configuration Item • Problem Report Status - Open Priority 1 and 2 by Type
		Component Status	2.3	• Design Progress With Replan • Subsystem Acceptance Status
		Action Item Status	2.4	• Action Item Status
	Incremental Capability	Increment Content - Components	2.5	• Incremental Content
Resources and Cost	Personnel	Effort	2.6	• Effort Allocation With Replan • Effort Allocation by Development Activity • Staffing Level
		Staff Experience	2.7	• Staff Experience
	Financial Performance	Earned Value	2.8	• Cost and Schedule Variance
		Cost	2.9	• Planned Cost Profile • Cost Profile With Actual Costs
	Environment and Support Resources	Resource Utilization	2.10	• Resource Utilization - Test Facilities
Product Size and Stability	Physical Size and Stability	Interfaces	2.11	• Interface Stability
		Lines of Code	2.12	• Software Size by Configuration Item • Software Size - Lines of Code
		Physical Dimensions	2.13	• Electrical Power Budget
	Functional Size and Stability	Requirements	2.14	• Requirements Stability • Requirements Stability by Type of Change
		Functional Change Workload	2.15	• Multiple Indicators for Change Requests • Change Requests by Priority

Common Issue Area	Measurement Category	Measure	Section	Indicator(s)
Product Quality	Functional Correctness	Defects	2.16	• Status of Severity 1 Defects • Defect Density • Defect Density in Code Inspections • Defect Classification Defects Configuration Item A • Defect Density Distribution
	Supportability – Maintainability	Time to Restore	2.17	• System Failures and Restorations • Mean Time to Repair or Fix • Mean Time to Restore System, With Threshold
		Cyclomatic Complexity	2.18	• Software Complexity - CI A • Software Complexity - CI A - Units With Complexity > 10
	Efficiency	Timing	2.19	• Response Time - On-Line Functions • Response Time During Test - On-Line Functions
	Portability	Standards Compliance	2.20	• Interface Compliance Validation
	Usability	Operator Errors	2.21	• Problem Reports by Type of Problem Data • Operator Error Distribution by Reason • Device Complexity Distribution
	Dependability - Reliability	Failures	2.22	• MTBF Ranges Based on Historical Data • Reliability Growth Tracked With Mean Time to Failure
Process Performance	Process Compliance	Reference Model Rating	2.23	• Reference Model Level - Continuous Type • Reference Model Level - Staged Type
		Process Audit Findings	2.24	• Process Audit Findings • Audit Findings by Reason Code
	Process Efficiency	Productivity	2.25	• Software Productivity - Historical versus Proposal • Evaluating Options Using Software Productivity
	Process Effectiveness	Defect Containment	2.26	• Requirements Defects Discovered After Requirements Phase

Common Issue Area	Measurement Category	Measure	Section	Indicator(s)
		Rework	2.27	• Development Effort by Activity - Compared to Total Rework Effort • Rework Effort - by Activity
Technology Effectiveness	Technology Suitability	Requirements Coverage	2.28	• Critical Technology Requirements • Technology Fit - Trends
	Impact	Technology Impact	2.29	• Mean Processing Time • Average Cost Per Picture • Estimated Yearly Maintenance Cost
	Technology Volatility	Baseline Changes	2.30	• Technical Volatility - Cumulative Releases • Technical Volatility - Emerging Technology • Technical Volatility - Established Technology
Customer Satisfaction	Customer Feedback	Survey Results	2.31	• Customer Satisfaction Survey
		Performance Rating	2.32	• Composite Performance Award Scores • Performance Award Category Scores
	Customer Support	Requests for Support	2.33	• Total Calls Per Month by Priority • Mean Response Time by Priority

Figure 5-11. Index of the Indicator Examples

2.1 Milestone Dates

Category: Milestone Performance

Common Issue Area: Schedule and Progress

Applicability: Applies to most types of projects

Analysis Guidance and Examples

During project planning and replanning, schedules derived from planned start and end dates should be analyzed. The analysis should determine whether any important activities or events are missing, whether each activity's planned start/end dates and duration are reasonable (based on other project assumptions such as staffing level or life cycle selected), and whether the overlap between activities is feasible.

During a replan, schedule slippage over time should be analyzed. Figure 5-12a is a Gantt chart used to compare multiple planning baselines for a single project. Each major project activity is included, with each successive plan represented by a different bar type.

In Figure 5-12a, Implementation ends slightly earlier in Plan 4 than in Plan 3, but Integration and Test finishes more than one month later in Plan 4 than in Plan 3. Given the extent of slippage that has already occurred on the project, the feasibility of meeting this new milestone must be evaluated.

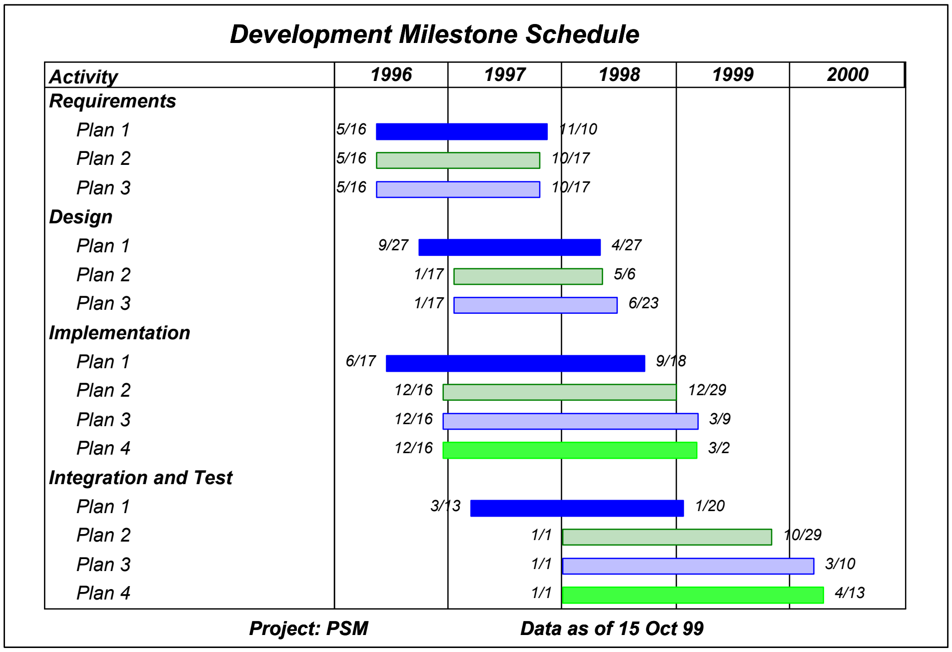

Figure 5-12a.

Throughout a project's life cycle, the Gantt chart is used to help identify the current status of major project events and to assess the impact of actual schedule slips on future activities and milestones.

In Figure 5-12b, planned and actual start and end dates show the status of the last four maintenance releases. The first three releases were completed, while Release 4 is still in progress. Both Releases 1 and 2 were completed late, and Release 4 is also projected with a late completion. For maintenance releases, late requirements changes often impact schedule and should be investigated.

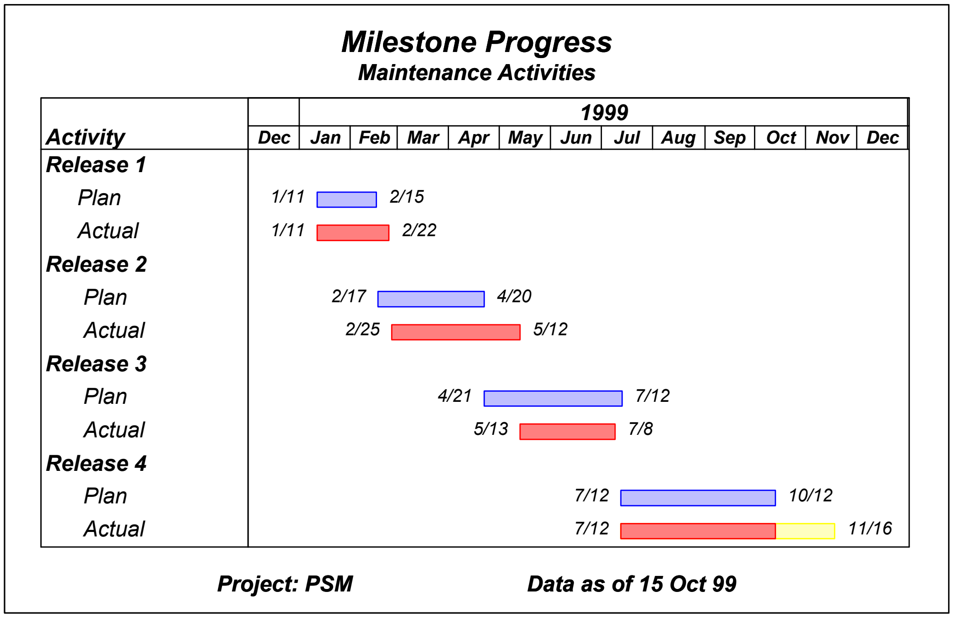

Figure 5-12b.

Additional Analysis

Further analysis of scope changes, staffing levels, work unit progress, and defect rates can uncover the reasons for schedule slips. The impact of schedule slips must be evaluated in light of project priorities and constraints.

Lessons Learned

Slips in critical path activities and events are of greatest concern because of the ripple effect on the project schedule. Schedules should contain a sufficient level of detail to monitor progress; multiple levels of detail may be needed to report progress to different audiences. If multiple increments or releases are planned, separate activities and events should be defined for each increment or release.

2.2 Problem Report Status

Category: Work Unit Progress

Common Issue Area: Schedule and Progress

Applicability: Applies to most types of projects

Analysis Guidance and Examples

Analyzing problem report data typically involves analyzing the number of problem reports written over time, their current status (open or closed), and other attributes (such as type or age). The quantity of problem reports indicates work (rework) effort and overall product quality. Closures rates help assess progress by indicating the amount of work or rework remaining.

A line chart (Figure 5-13a) shows both the cumulative number of problem reports and the number of problem reports closed to date. The difference represents the total number of problem reports that are still open. Figure 5-13a indicates that a large number of new problems have been discovered over the past year. However, in the past several months, the reporting rate has tapered off. While the closure rate has not kept pace with the reporting rate, the number of open problem reports is shrinking, as problems are steadily being closed, and fewer new ones are being reported.

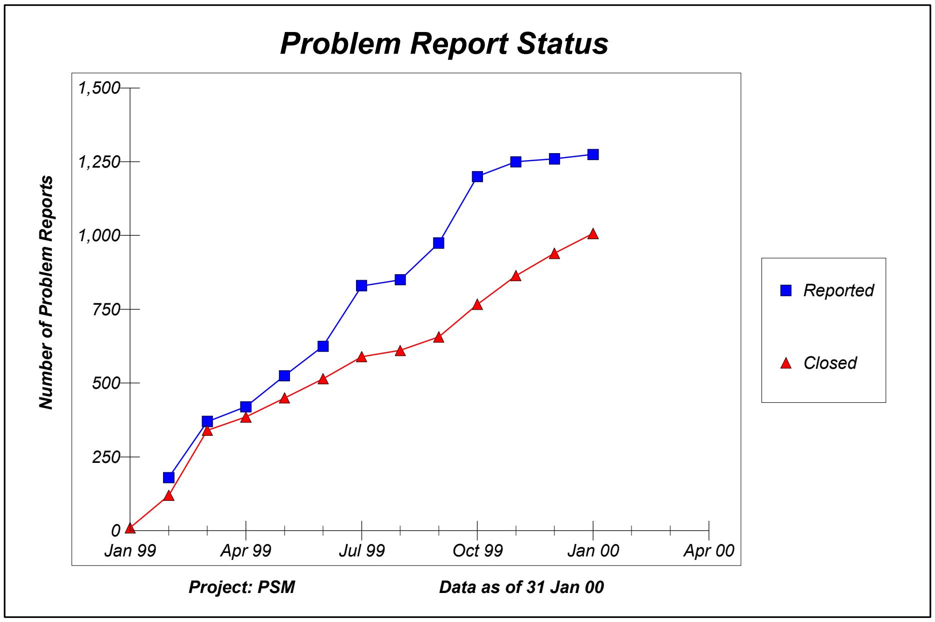

Figure 5-13a.

To learn more about the remaining open problem reports, additional analyses were performed. The bar chart in Figure 5-13b includes all open problem reports divided into categories by age. This was done by calculating the number of days elapsed since the problem was reported and grouping the problem reports by age.

Figure 5-13b shows an average open age of 5.7 weeks. This average is below the desired maximum of eight weeks. The maximum age of problems is determined by considering the length of the project, the project's current status, delivery requirements, and the type and severity of defects discovered.

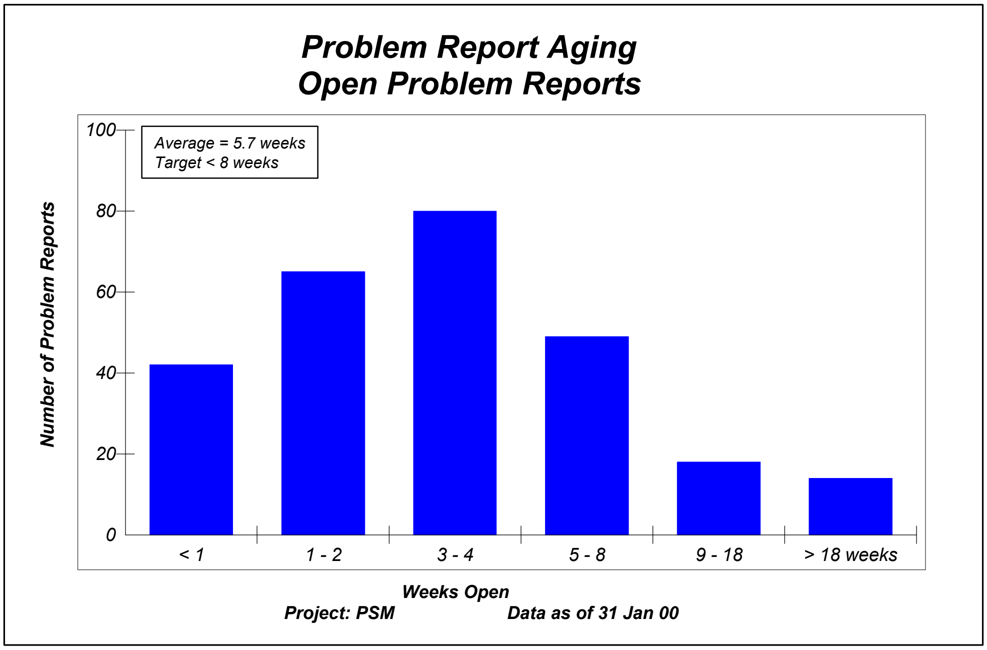

Figure 5-13b.

Another graph shows the same open problem reports by priority code (Figure 5-13c). This graph reveals that over half of the open problem reports are either priority 1 or 2.

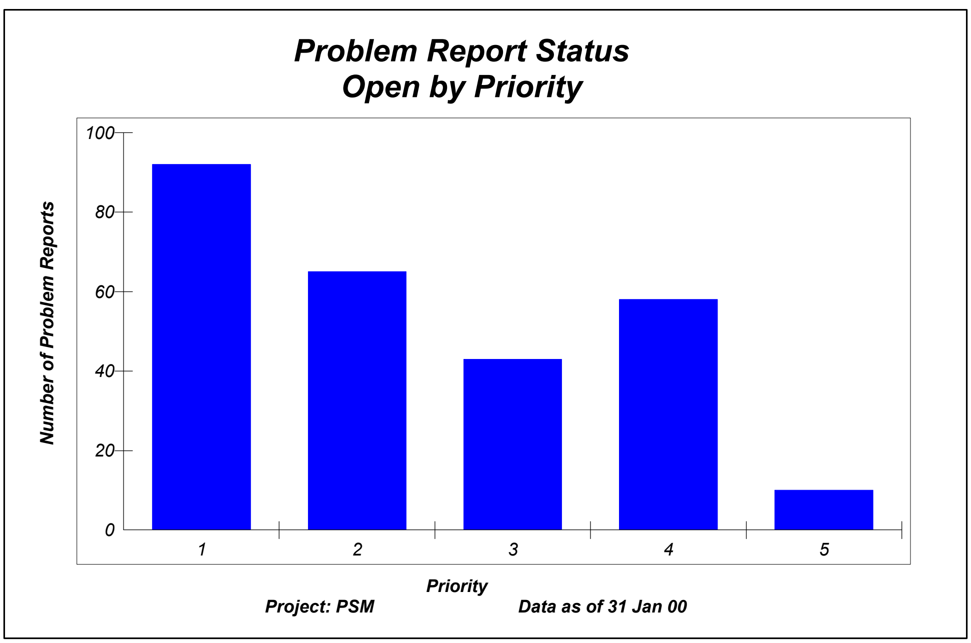

Figure 5-13c.

Because priority codes 1 and 2 represent significant problems, other types of analyses were performed to assess the impact of the remaining work/rework (Figures 5-13d and e below).

Figure 5-13d illustrates a breakout of priority 1 and 2 problem reports by configuration item (CI). There is no major concentration of problems associated with a particular CI.

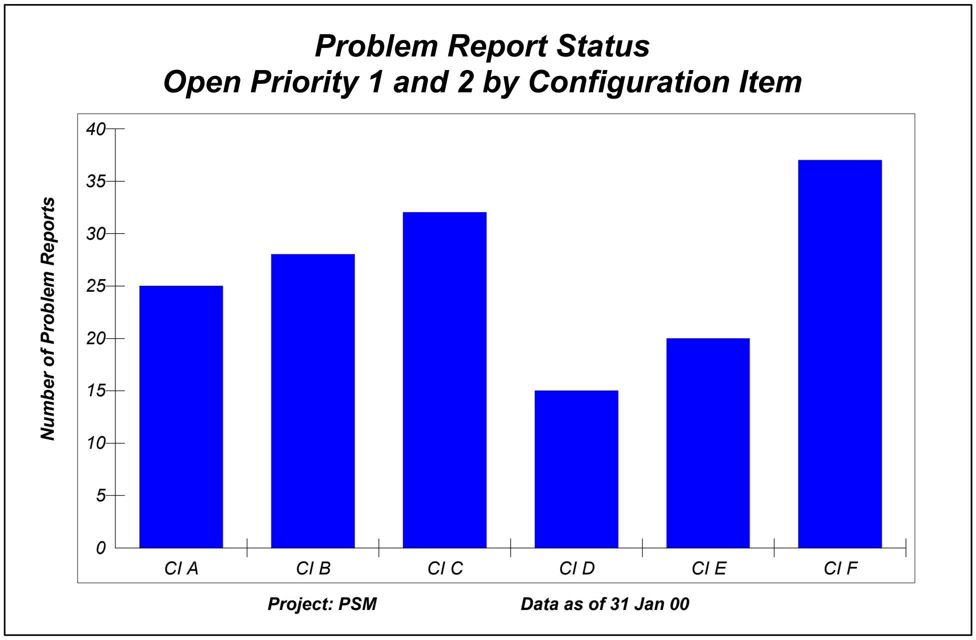

Figure 5-13d.

Figure 5-13e is a breakout of the same priority 1 and 2 problem reports by problem type: performance, logic, interface, and other. This graph indicates that many of the open, high-priority problems are related to performance - a major concern, since performance problems are often difficult to fix and may require redesign of major pieces of the system.

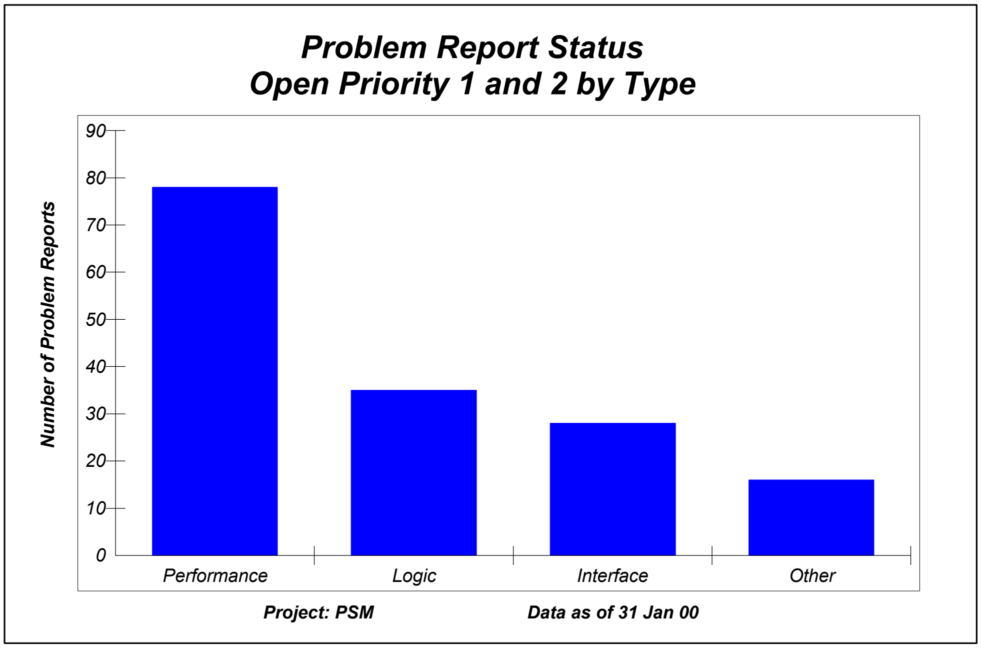

Figure 5-13e.

Additional Analysis

The current level (stage) of testing and test progress should be evaluated to determine why problem reports have recently decreased. If the tests are being completed successfully and the project is nearing completion, this is a good sign. If testing has prematurely slowed or halted, this may indicate a significant problem. Additionally, problem reports that have been open for a long period should be investigated. Open problem reports may no longer be pertinent, or may represent major problems that require substantial rework and special management attention to resolve.

Lessons Learned

The closure rate should remain close to the reporting rate throughout the project. Large gaps between the two trend lines indicate that problem correction is being deferred, which could result in serious schedule, staffing, and cost problems later in the project. A flat problem reporting trend line during design, code, or test may indicate that peer reviews and tests are not being performed. Open problem reports should be monitored by priority to insure that high-priority defects are being fixed first.

During testing, deferring problem correction can impact test progress by making it impossible to proceed past problem areas and complicating regression testing.

During operations and maintenance, the age of customer problem reports should be monitored to ensure timely resolution. Analyzing problem report age and priority may indicate that high-priority problems (that may be more difficult to resolve) are being deferred longer than low-priority problems.

2.3 Component Status

Category: Work Unit Progress

Common Issue Area: Schedule and Progress

Applicability: Applies to most types of projects

Analysis Guidance and Examples

Analyzing component status helps identify or predict schedule slips by comparing the number of work units or components completing a project phase to the number planned for completion to date. In the example in Figure 5-14, design progress is graphed with a line chart depicting cumulative measures for the original plan (Plan 1), the current plan (Plan 2), and the actual components designed to date. Each point is calculated by adding the number of components allocated for the reporting period to the corresponding cumulative total from the last reporting period. The figure shows that design progress was behind the original plan at the end of August 1999, resulting in a replan of the overall activity. Actual design progress has remained fairly close to the new plan (Plan 2). The plan line, however, requires a significant increase in the completion rate over the next few months, raising concern about the feasibility of the plan.

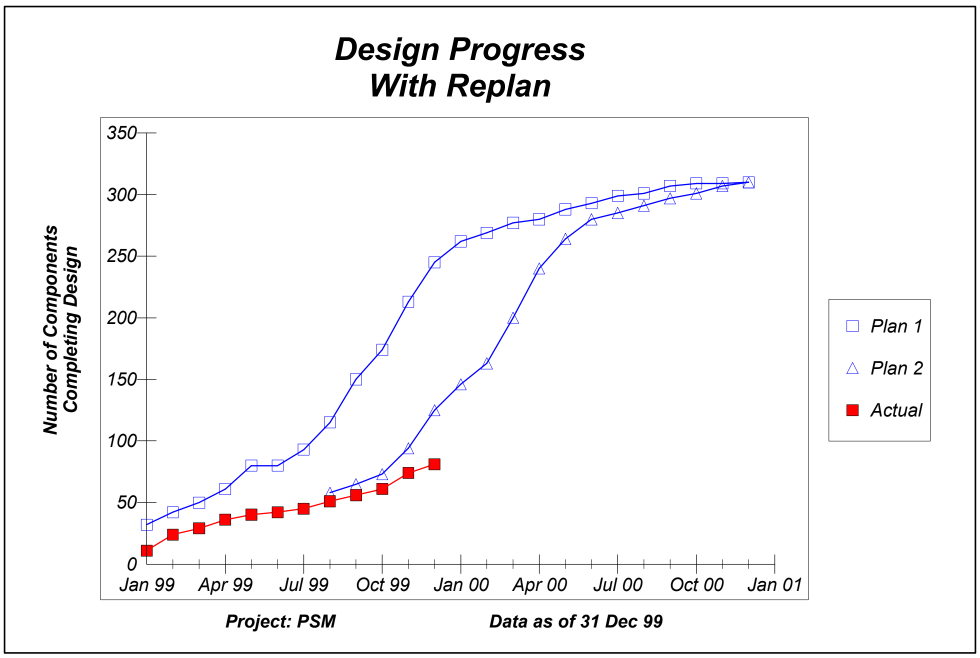

Figure 5-14.

Major changes in the rate of progress should be investigated. Once an actual trend line is established, it is difficult to modify the rate of completion. A 10-percent cumulative deviation, or 20-percent-per-period deviation from the plan usually is viewed as significant.

Additional Analysis

A more detailed analysis is usually required when actual progress lags behind planned progress. For example, an additional indicator showing progress by subsystem may identify which components are most behind. Staffing levels, experience levels, changes in scope, and quality problems may all contribute to a lack of progress and should be investigated.

Lessons Learned

To accurately assess component status, measures must be based on objective exit criteria (such as checking a component design into the configuration management library). Criteria should be documented.

2.4 Action Item Status

Category: Work Unit Progress

Common Issue Area: Schedule and Progress

Applicability: Applies to most types of projects

Analysis Guidance and Examples

This indicator shows relative progress in working through action items. Items are typically tracked by their various states or by the actions performed on them. For example, an action item can be in an open or closed state.

An Action Item Status indicator is illustrated below. Figure 5-15 plots counts of action items completed or closed during each period. This includes items opened and closed during the current period as well as items opened during a previous period but closed during this period. In this example, the number of new items introduced each period has peaked and is now shrinking. Since closures are relatively steady, the backlog of unresolved actions is declining.

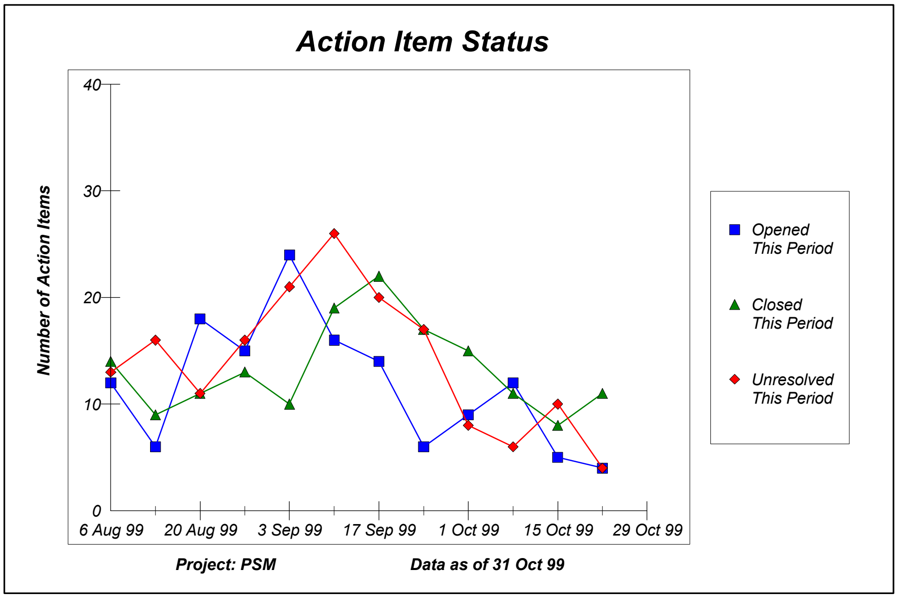

Figure 5-15.

Additional Analysis

If there are more than two states (e.g., open, resolution selected, closed), examine the proportion of tracked items in each state to determine where to focus attention. For example, if most actions have resolution plans in place, the problem may be completing the resolution plans. If most issues lack resolution plans, focus on developing the plans.

Lessons Learned

When defining the states of tracked items, definitions must be discrete, unambiguous, and mutually exclusive so that they can easily be tallied.

2.5 Increment Content - Components

Category: Incremental Capability

Common Issue Area: Schedule and Progress

Applicability: Applies to most types of projects

Analysis Guidance and Examples

When multiple increments or releases are planned, this indicator helps to determine if the schedule is realistic and to monitor changes (often, functionality is deferred to later increments).

During planning and replanning activities, evaluate the feasibility of the increment plan. Consider the allocation of components to increments in terms of overlapping work effort and the likelihood of slippage. The sum of components allocated to all increments should equal the total number of components scheduled for the final release.

The bar chart in Figure 5-16 summarizes the number of software units for each build (software increments often are referred to as builds). Plan 1 shows units that were originally planned for delivery; Plan 2 shows units planned for delivery in a second build plan; and Plan 3 shows components planned for delivery in the latest build plan (which is based on the actual number of units delivered in Builds 1 and 2).

Figure 5-16 shows that units in both Builds 1 and 2 were deferred to Build 3, increasing its size by over 30 percent. Build 3 is now significantly larger than either of the previous builds. These deferments will probably result in implementation and test delays for that build, possibly impacting customer delivery milestones.

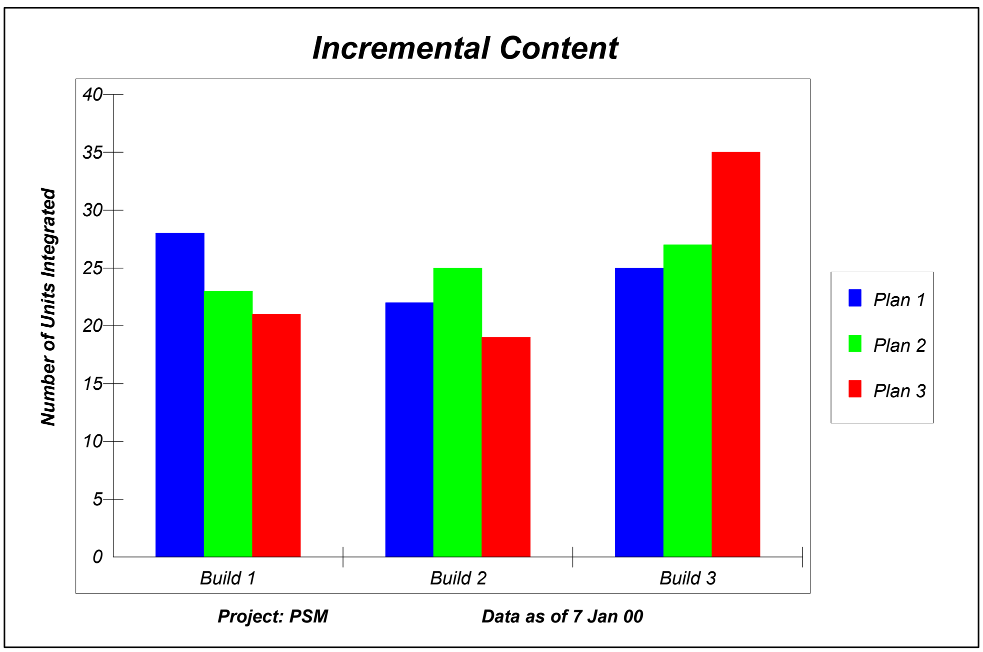

Figure 5-16.

Additional Analysis

Schedule and progress data should be evaluated when functionality is deferred to later increments. Effort plans should be compared to schedule plans to determine whether sufficient resources and time have been allocated for overlaps and different increment sizes.

Lessons Learned

Deferments to later increments without adjustments to the schedule are risky. A 10-percent variance in a single increment, or a cumulative 20-percent variance across two or more increments should be considered significant.

2.6 Effort

Category: Personnel

Common Issue Area: Resources and Cost

Applicability: Applies to most types of projects

Analysis Guidance and Examples

Effort indicators should be analyzed throughout a project to assess the adequacy of planned effort and to track the actual allocation of labor to development activities. Two examples are shown below. In Figure 5-17a, the latest plan (Plan 2) is compared to the original plan (Plan 1) and to the actual effort expended to date. While Plan 2 appears more realistic (because it is more consistent with actual effort allocation to date), the acceptability of extending the schedule by several months must be determined. Also consider whether the new plan calls for additional effort overall. In this example, total effort has not increased; otherwise, the impact on project costs and the availability of additional resources should be considered.

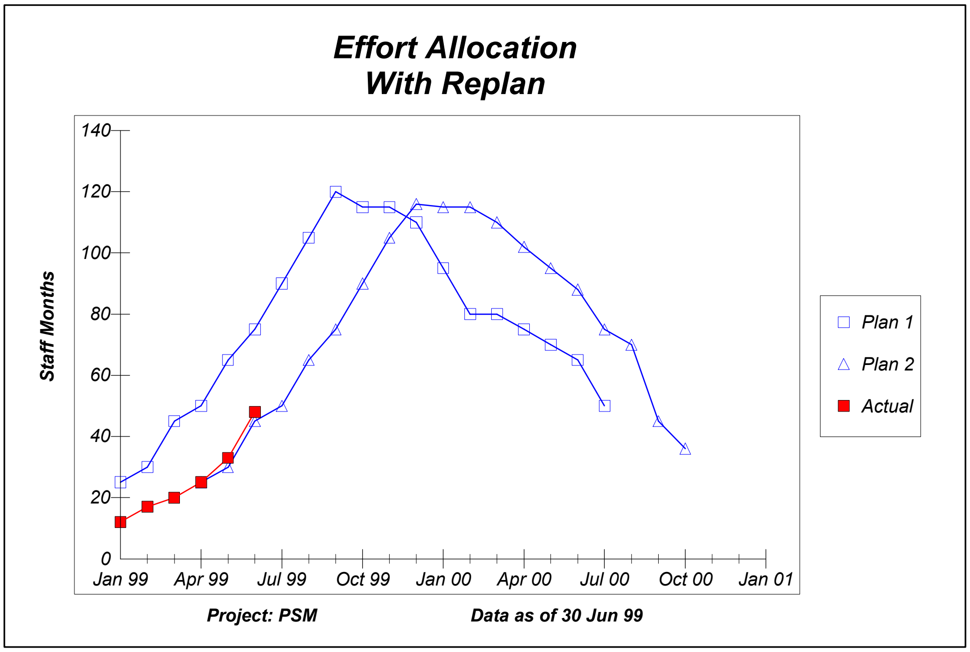

Figure 5-17a.

On this project, the distribution of effort between the project's life-cycle phases was checked for feasibility (Figure 5-17b). The planned effort distribution should be compared to historical data from similar past projects (when available) to verify that the planned distribution is realistic.

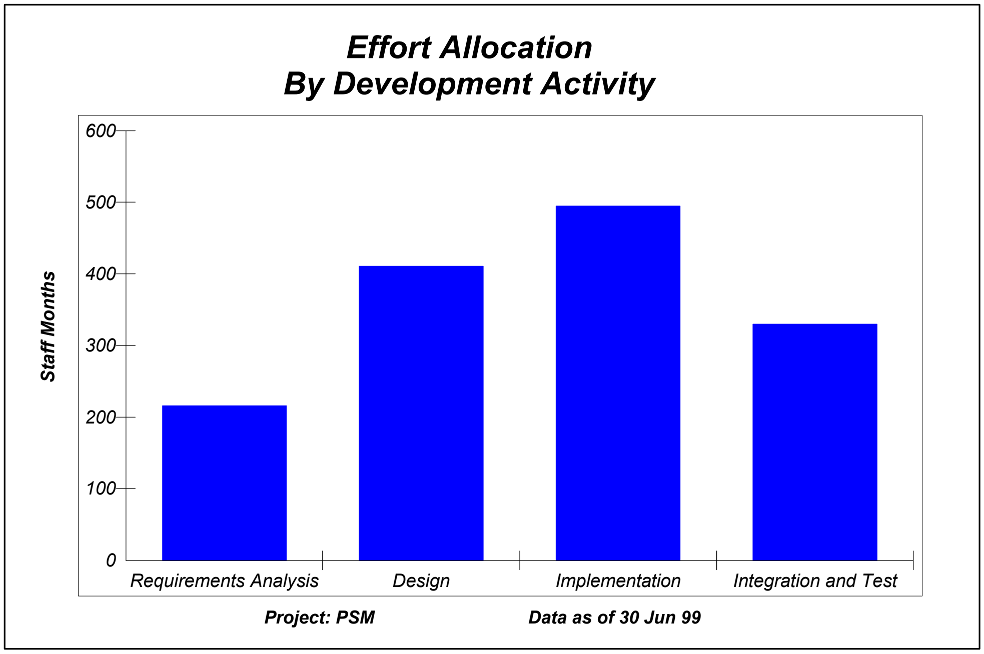

Figure 5-17b.

Regular monitoring of effort expenditures may indicate possible schedule slips and resource availability issues early on. Track the actual effort by increments, iterations, or phases to ensure that sufficient time is being spent in early project phases. The impact of this problem could be significant, requiring additional time in later phases to address missing requirements and to fix defects. Quality problems and schedule slips may also arise.

Figure 5-17c tracks effort on a maintenance project with a fixed staffing level (level-of-effort project). While the fixed staffing level was incorporated into project plans, actual effort expended to date did not achieve this level. In March and April, several members of the project were loaned to another project. In May, some new hires were assigned to maintenance as a training opportunity. Due to summer vacations, less time was spent in July than planned.

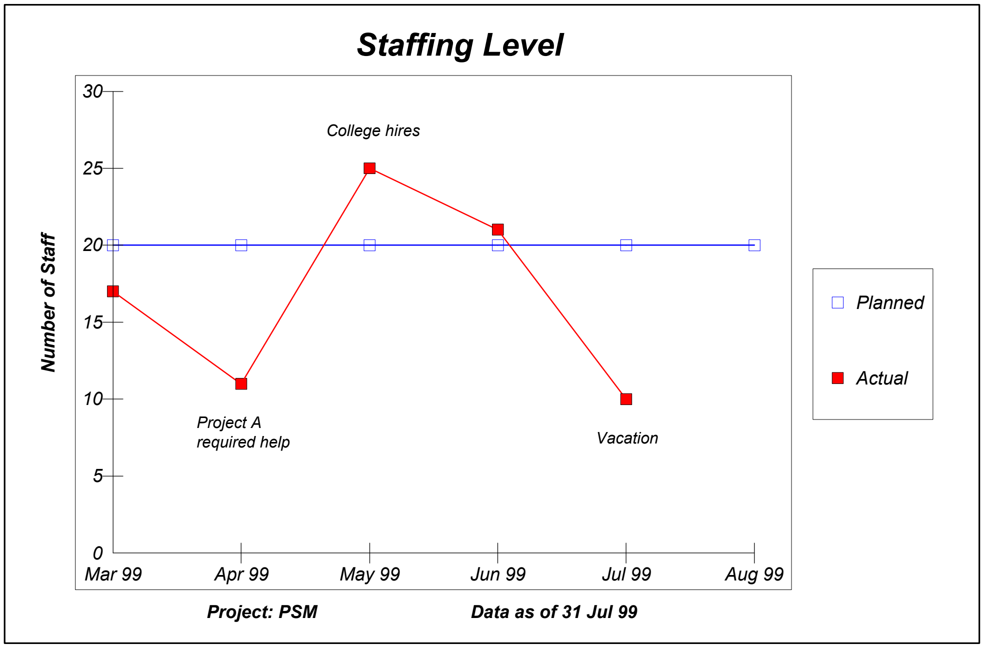

Figure 5-17c.

Additional Analysis

Consider the scope of work to be performed, staff experience, task assignments, schedules, quality problems, and rework when analyzing the adequacy of effort and when exploring why effort may not be tracking to plans.

Lessons Learned

Watch for large changes in the application of effort. It is difficult to add many people effectively to a project within a short period of time. Large overruns during integration and test may indicate software quality problems, significant defects, and possible completion delays.

Staffing levels may be below plan because the original plan was unrealistic. Vacations, holidays, or other events may not have been considered, or non-project activities may be draining resources from the project. The latter is often a problem in operations and maintenance because the support team is often responsible for enhancing and releasing new versions of the system, as well as fixing problems with the fielded version.

2.7 Staff Experience

Category: Personnel

Common Issue Area: Resources and Cost

Applicability: Applies to most types of projects

Analysis Guidance and Examples

Staff Experience is analyzed to assess whether the personnel assigned to the project possess the experience necessary to develop a system that meets customer needs. As an example, Figure 5-18 compares the supplier's staff years of real-time distributed systems experience (initial and current) to contract requirements. The histogram bars were produced by sorting the staff's experience by the real-time distributed systems experience, and summing the number of staff in six experience categories. This distribution of experience levels could then be compared to contract requirements. Since contract requirements specify an average of three years real-time distributed systems experience, the staff's current experience level was also calculated and displayed on the graph.

Figure 5-18 shows that the supplier started the project with a staff average of 3.4 years of real-time distributed systems experience. To further investigate recent schedule slippage and low productivity, updated staff experience data was requested. The new data reveals that, while staff size has remained constant (in spite of turnover), the experience levels of replacement staff have dropped. The average experience is now only 2.4 years. Additional analysis should be performed of skill requirements and staff allocations for the remaining tasks.

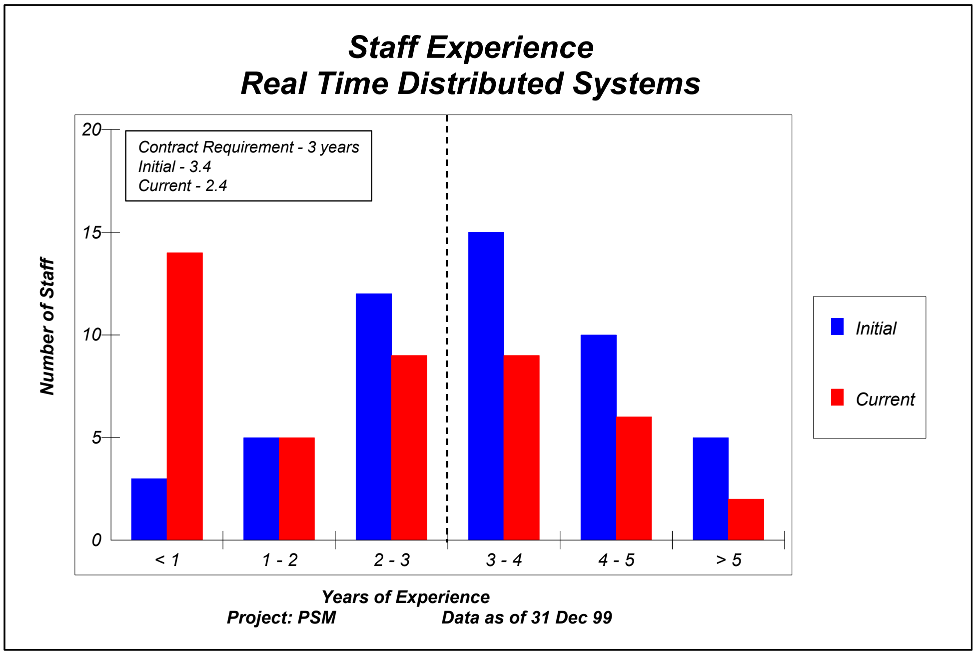

Figure 5-18.

Additional Analysis

Critical skills should be identified at the start of a project. Evaluate the availability of appropriate skills for each task.

Lessons Learned

Analysis of staff experience is usually performed at major milestones on large projects, unless other analyses point to a staffing problem. Data on years of experience for critical skills should be kept up to date. Evaluate staff experience data with respect to project performance.

2.8 Earned Value

Category: Financial Performance

Common Issue Area: Resources and Cost

Applicability: Applies to most types of projects

Analysis Guidance and Examples

Earned Value places all activities into time-phased increments that establish a cost and schedule baseline. Earned value provides an indication of cost and schedule performance, based on dollars budgeted per work activity. The activities are typically aggregated to a project or system level using an established work breakdown structure (WBS) to indicate earned value. Earned value uses three primary data elements: Budgeted Cost of Work Scheduled (BCWS), Budgeted Cost of Work Performed (BCWP), and Actual Cost of Work Performed (ACWP).

Analysis of earned value provides insight into the project's ability to complete scheduled activities on time and within cost, and indicates whether the project is ahead or behind of planned funding for work performed.

Earned value indicators usually are represented with a line chart. Figure 5-19 shows a plot of performance in terms of percentage variance from plan. The two measures plotted on the graph were calculated as:

Schedule Variance = (BCWP - BCWS) / BCWS

Cost Variance = (BCWP - ACWP) / ACWP

Results near zero indicate that the project is proceeding according to plan. Negative results are an indication that the project is behind schedule or over budgeted cost. Positive results indicate the project is ahead of schedule or under budgeted cost. Thresholds of +/- 10% commonly are used to trigger corrective action.

Alternatively, the cumulative ratio of budgeted to actual values can be computed as follows:

Schedule Performance Index = BCWP / BCWS

Cost Performance Index = BCWP / ACWP

Plotting these measures yields a target value of 1.0 when performance matches the plan.

Figure 5-19 indicates that the project has been behind schedule and over cost from early in the project. The large spike in schedule variance in August should have been investigated. It may be related to either actual performance improvement or to the accounting process. Performance for the last three months is outside thresholds, indicating that the project is seriously behind schedule. Replanning is necessary.

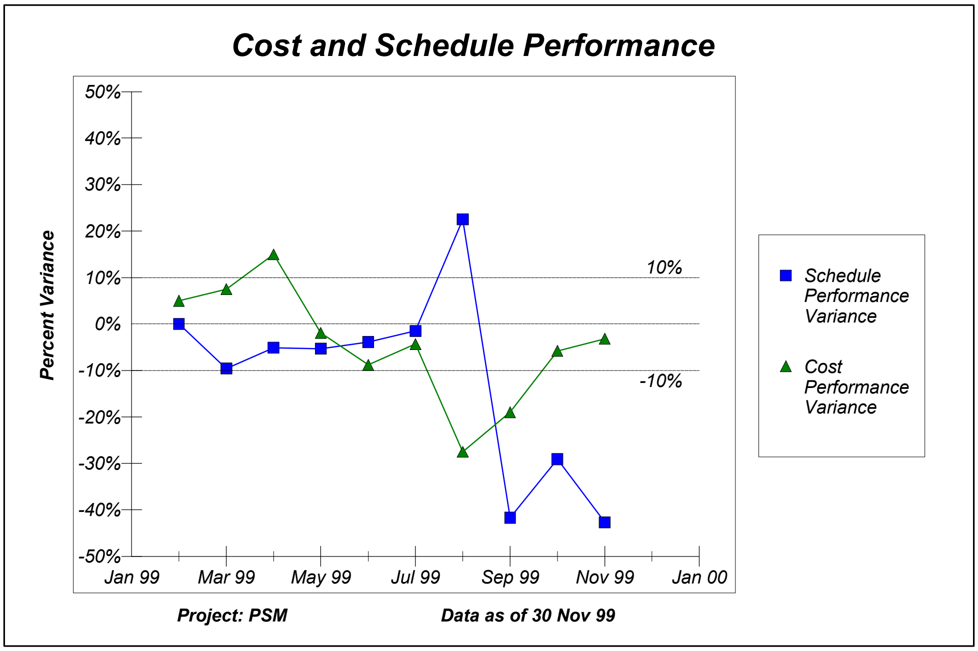

Figure 5-19.

Additional Analysis

During feasibility analysis, make sure that the BCWS and BCWP lines agree with other plans. A fast rise may reflect large equipment purchases or a sudden increase in staffing.

During the project, additional analysis using other measures may be needed to identify the underlying cause of schedule and cost problems.

Lessons Learned

Large cost or schedule variances should be investigated as soon as possible to identify and correct the problem. If the financial performance baseline is replanned, keep the previous data to justify the change.

2.9 Cost

Category: Financial Performance

Common Issue Area: Resources and Cost

Applicability: Applies to most types of projects

Analysis Guidance and Examples

This indicator compares planned costs to actual costs. It helps to assess whether a project is performing within cost constraints. In contractual situations funding may be provided in increments. The availability of funding must be considered in developing a spending plan.

During the planning stage, use a line chart (Figure 5-20a) to plot the cumulative spending plan, planned funding increments, and total cost budget. Spending above the funding plan indicates work at risk. The figure shows a potential funding problem in March 1999. Delaying some activities would bring the spending plan back in line with funding.

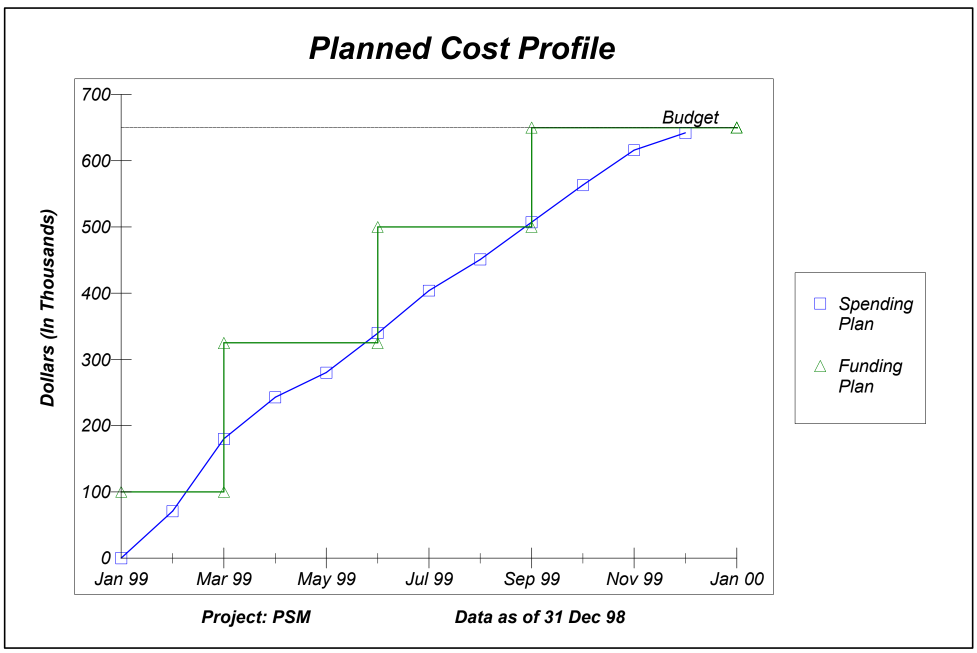

Figure 5-20a.

The spending plan must be realistic over the period of performance. Large changes in the spending rate may indicate purchases or staffing changes that need to be monitored. Figure 5-20a shows a relatively consistent planned expenditure rate.

During project performance, actual costs should be tracked against the plan. Figure 5-20b shows that actuals were somewhat below plan, but are now close to planned costs.

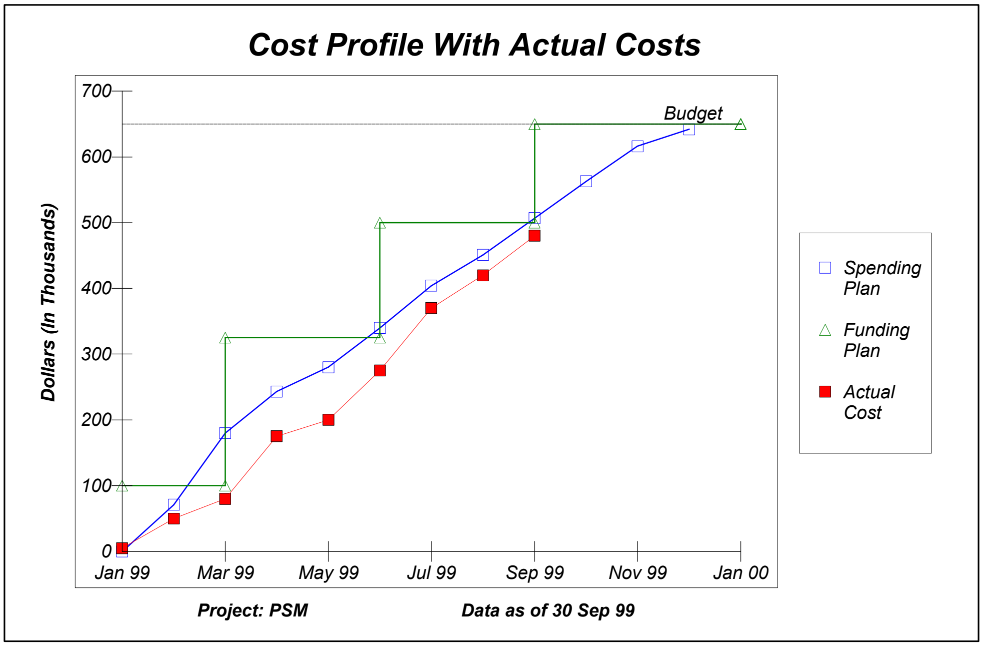

Figure 5-20b.

Additional Analysis

Cost analysis should be coordinated with effort profiles and schedules. Ask these questions about cost variances and overruns:

- Are overruns due to activities costing more than planned?
- Are all activities costing more or only certain types?
- Is work beginning ahead of schedule?
- Is productivity higher or lower than expected?

Lessons Learned

In a labor-intensive project, cost data should track closely to effort data.

2.10 Resource Utilization

Category: Environment and Support Resources

Common Issue Area: Resources and Cost

Applicability: Applies to most types of projects

Analysis Guidance and Examples

This indicator helps determine whether project resources are available and being utilized according to plan.

For example, Figure 5-21 compares test facility utilization to planned availability. The figure graphs three measures:

1. Actual test facility availability (total time minus maintenance downtime)
2. Scheduled project utilization
3. Actual project utilization (hours logged)

Analysis of Figure 5-21 shows that testing at the facility started one month late. Also, a September shortfall in facility availability may have impacted progress that month. Since the actual hours used to date are significantly below planned, a replan should be considered, in case additional testing time is needed. In addition, the cause of the shortfall should be investigated to plan for future availability.

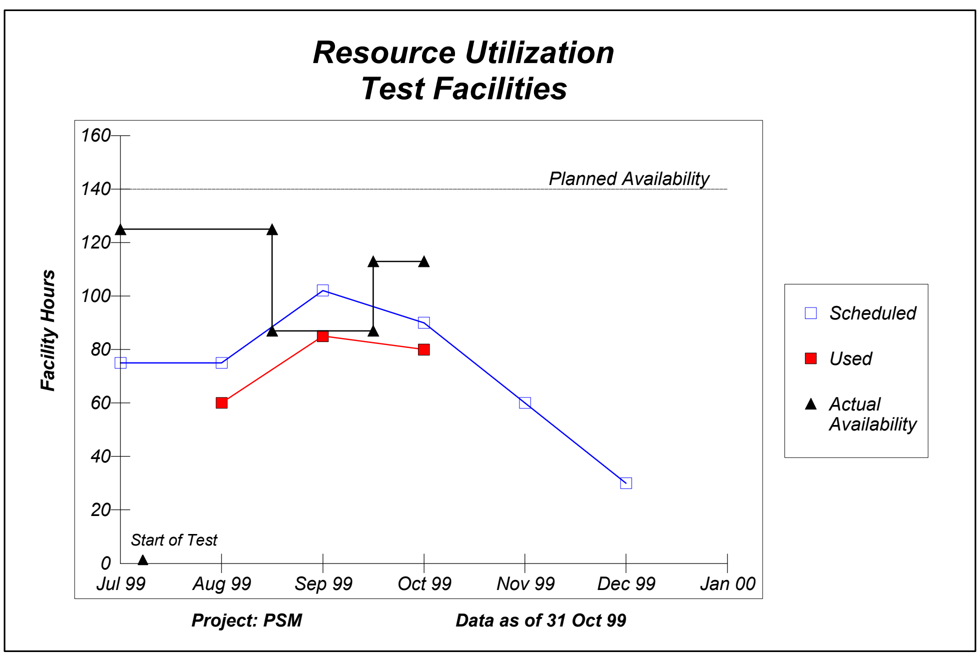

Figure 5-21.

Additional Analysis

The analysis should consider problems in scheduling the test facility resources if testing is delayed. Also, testing progress to date should be reviewed for a more complete analysis.

Lessons Learned

Unexpected variances in resource utilization or availability should be investigated to prevent future problems.

2.11 Interfaces

Category: Physical Size and Stability

Common Issue Area: Product Size

Applicability: Applies to both software and systems engineering

Analysis Guidance and Examples

The Interfaces indicator helps assess progress in defining system interfaces and in stabilizing their definitions over time.

Figure 5-22 provides an example in the form of a stacked column chart. The column segments represent different interface states, including:

- Interfaces that were newly added in the current month
- Interface definitions that have changed since the previous month
- Interfaces that have not changed from the previous month

The total height of a column represents the total number of interfaces identified for the system at a given time. If any interfaces are deleted, the total number of interfaces is reduced. Significant project milestones such as PDR and CDR are indicated to provide context to the chart.

Figure 5-22 shows that the number of interfaces defined have stabilized in the last few months and that the interface specifications themselves have stabilized (few changes have occurred in the last few months).

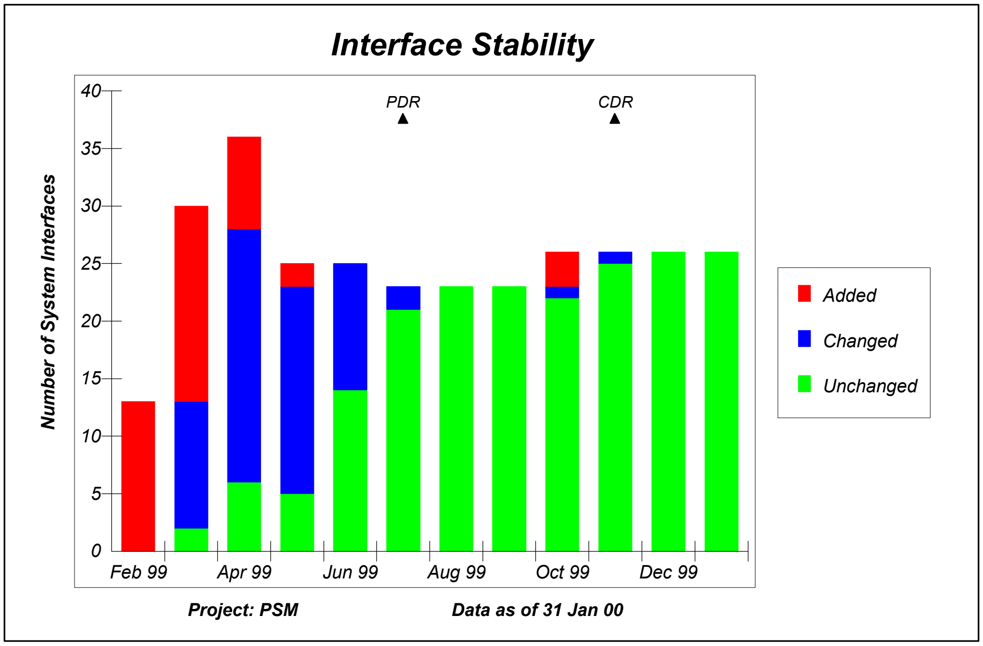

Figure 5-22.

Additional Analysis

Interface size and stability should be compared with measures of requirements growth and stability. Focus attention on interface definitions that cannot be completed early because of dependencies on systems or products external to the project. Also consider architectural optimization tasks that may reduce the number of interfaces.

Lessons Learned

A good deal of volatility in interface definitions can be expected while the architecture is under development. During architecture definition, the analysis should focus on the total number of interfaces, with the goal of minimizing interfaces whenever possible. Once the architecture has stabilized, the analysis should focus on minimizing the number and trend of interface *changes*.

It is important to show key project milestones on the indicator chart to identify the potential risks of late interface changes.

The number of interfaces specified each month should be preplanned so that progress can be tracked against the plan.

2.12 Lines of Code

Category: Physical Size and Stability

Common Issue Area: Product Size and Stability

Applicability: Applies to software engineering

Analysis Guidance and Examples

Lines of Code is often the primary size measure for a software-intensive project. It is used to estimate software development effort and to prepare project schedules. Therefore, it is important to review lines of code estimates every time a software project is replanned. Whenever possible, lines of code estimates should be compared with actual software size data from similar, completed projects.

A bar chart (Figure 5-23a) is used to show size estimates for a project by its Configuration Items (CIs). (Size is often analyzed at this level because CI-level estimates can be compared to historical data from CIs with similar functionality.) The chart shows an overall increase in size during the last two replans. Most of the estimated size increase can be attributed to CI C. Additional effort and schedule may now be required to complete this CI's development activities.

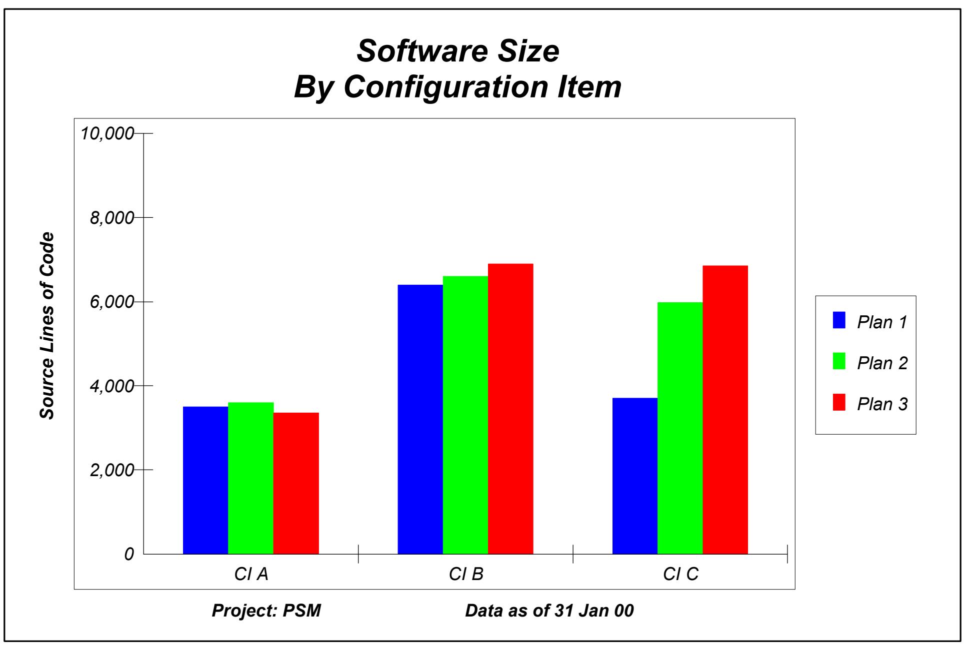

Figure 5-23a.

During project execution, unplanned code additions and changes can adversely influence schedules and costs. Indicators can monitor progress by comparing the amount of code that is actually developed and modified over time to plans for code development and growth. For example, a line chart (Figure 5-23b) compares actual size growth to target and threshold as development proceeds. The threshold is based on historical data showing that 20% size growth typically can be accommodated without replanning.

Figure 5-23b shows that actual code production exceeded the size estimate and the upper threshold. The cause of the code growth was determined to be new requirements identified during initial testing. Resource allocations, schedules, budgets, test schedules and plans were impacted by this unexpected size growth.

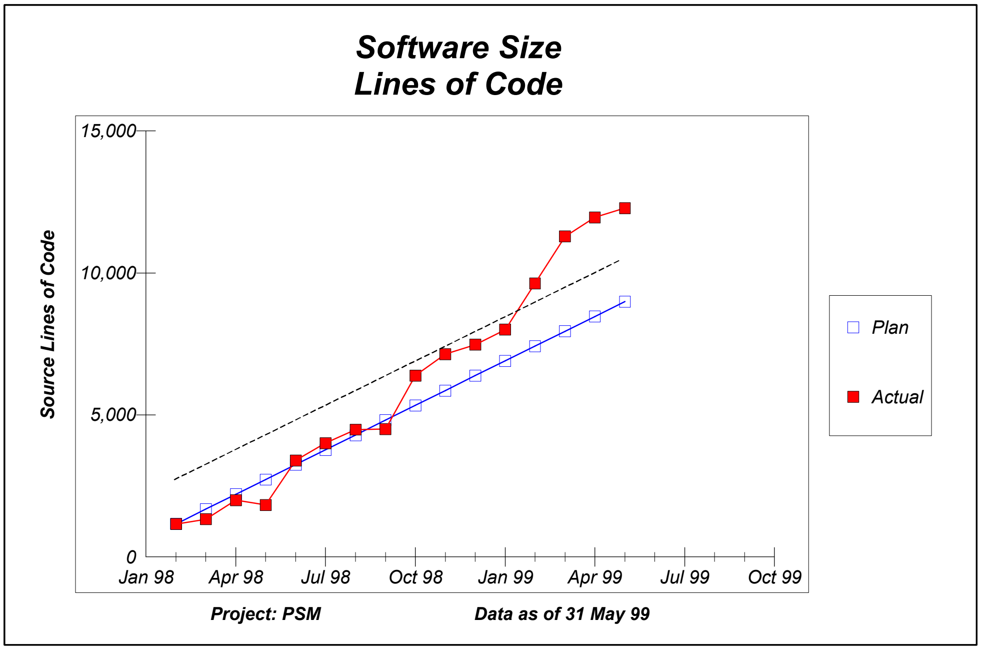

Figure 5-23b.

Additional Analysis

Many projects develop software using several languages. Also, many projects use supplied code libraries, reusable code, generated code, and modified code. All these sources should be considered when analyzing code growth and stability.

Size estimates should be monitored over time along with development staffing profiles. Sufficient staff should be assigned during each time period to complete coding assignments. Resource allocation should consider types of code, level of rework, concurrent assignments, non-project time, and programmer productivity.

Lessons Learned

It is not unusual for moderate increases to occur in total software size - up to 20 percent, even with fairly stable requirements. Larger increases in estimates or actuals should be investigated.

2.13 Physical Dimensions

Category: Physical Size and Stability

Common Issue Area: Product Size

Applicability: Applies to systems engineering

Analysis Guidance and Examples

Systems may need to fit or operate within physical constraints in order to achieve their purpose. For example, electrical power consumption is a physical characteristic. The system under development may have a maximum capacity in terms of power available. Power budget indicators help to ensure that power needed by system components will not exceed available power. When power demands exceed the specification, corrective actions must be taken, such as raising the specification (add more power generation), reducing the power consumption of components (perhaps only at worst-case scenarios), or reducing the power margins (provided actual data has become available). Historical charts from similar projects can help establish necessary power margins.

Figure 5-24 shows an example of tracking power consumption estimates over time. The rise in power reflects modifications to the design and integration data from components actually consuming power. As more detailed estimates and actual power draws for components become available, the estimate accuracy increases. A design margin is added to the chart for each estimate/actual data point. The margin is reduced as the design matures and actual power consumption is measured.

Figure 5-24 shows a significant increase in total power consumption between September 1999 and October 1999. Such an increase needs to be explained. Potential causes include the addition of components, transmission loss, or the cumulative effects of components exceeding their specifications. Additional analyses should be performed to determine the source of the problem.

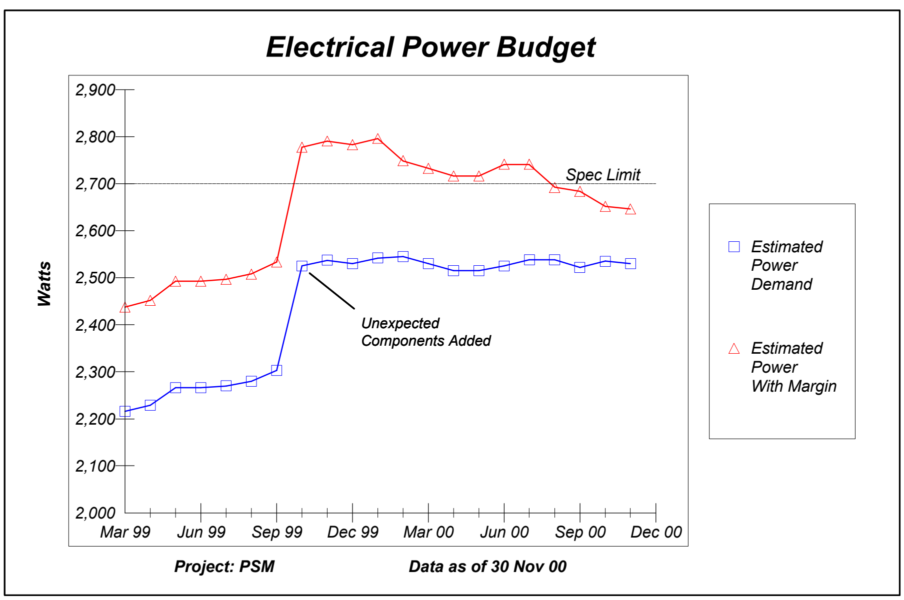

Figure 5-24.

Additional Analysis

The power budget for each subsystem and component should be tracked at a sufficient level of detail to isolate the problems. In addition, various states and modes of operation must be examined periodically to be sure that the "worst case scenario" can be accommodated.

Lessons Learned

Use similar charts for other physical dimensions, such as thermal budgets, weight budgets, physical dimensions, pointing accuracy, and signal strength. Keep track of when data was entered, the source, and the fidelity of the data (such as estimate, calculation, test results, or actual measurement). Error margins may be calculated for each component based on a standard number for each error category, or the customer may dictate margins. Error margins should only be reduced by negotiation.

2.14 Requirements

Category: Functional Size and Stability

Common Issue Area: Product Size and Stability

Applicability: Applies to most types of projects

Analysis Guidance and Examples

The Requirements indicator provides insight into changes to requirements throughout a project's life. It serves as a leading indicator of size growth, schedule delays, cost increases, and rework.

For example, the line chart in Figure 5-25a shows two pieces of requirements-related information. The top line is the trend in the total number of requirements defined to date. The bottom line represents the total number of changes made each month (the number of requirements added, changed, and deleted during the month).

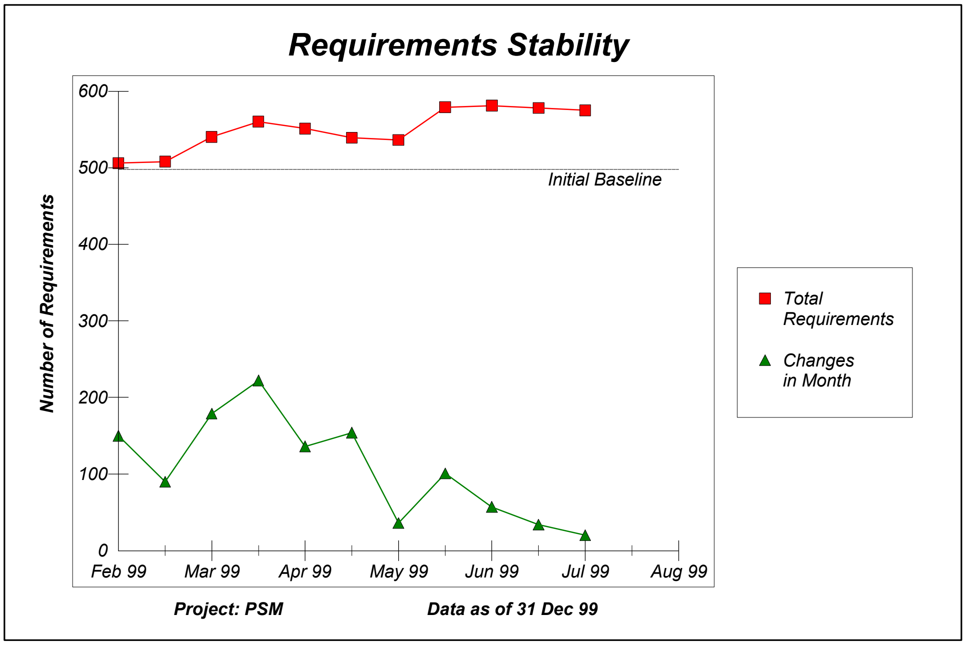

Figure 5-25a.

Figure 5-25a shows an increase in requirements after the March milestone review, which was expected. An unexpected increase in requirements in September can be traced to the review held in August.

A bar chart (Figure 5-25b) provides more detail about whether the changes were additions, modifications, or deletions.

In Figure 5-25b requirements changes in April, May, and June resulted from the March review and were probably expected. The substantial changes in September, driven by the August review, were cause for concern because of the magnitude and timing. Approximately 20 percent of the total requirements were affected during this last period, and total requirements increased by almost 10 percent. The added and changed requirements needed further development and did not stabilize for several months following Review 2. The timing of the requirements changes is a problem because the project is already well into its design phase.

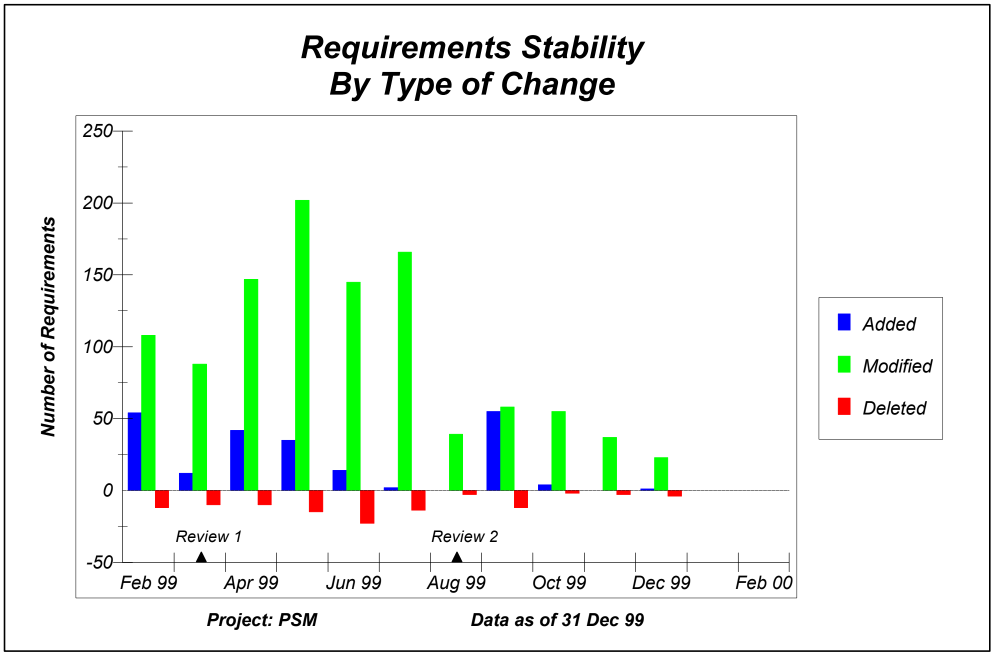

Figure 5-25b.

Additional Analysis

Consider the types of additions and changes to requirements as well as the reasons for the changes in conducting more detailed analyses. Requirements changes can be classified by area affected (such as system interface, user interface, performance, or key function) or by cause (such as missing, unclear, incomplete, or incorrect requirements).

A high level of requirements volatility may require adjustment to current resource allocations, effort estimates, budgets, and schedules.

Lessons Learned

Constantly changing requirements, or a large number of additions after a requirements review, can lead to schedule and budget problems later in the project. Requirements should be tracked at a more detailed level (such as by CI) to isolate the source of frequent changes.

2.15 Functional Change Workload

Category: Functional Size and Stability

Common Issue Area: Product Size and Stability

Applicability: Applies to most types of projects

Analysis Guidance and Examples

Function Change Workload indicators provide insight into the stability of the system being supported and enhanced.

For example, Figure 5-26a includes multiple graphs tracing software and system change requests (new or changed requirements) through the stages of completion from submission to incorporation into a product release. The graphs 1) identify bottlenecks in completing change requests, 2) predict the size and timing of upcoming workloads, and 3) identify areas where scheduling adjustments or process changes are needed to smooth out the workload.

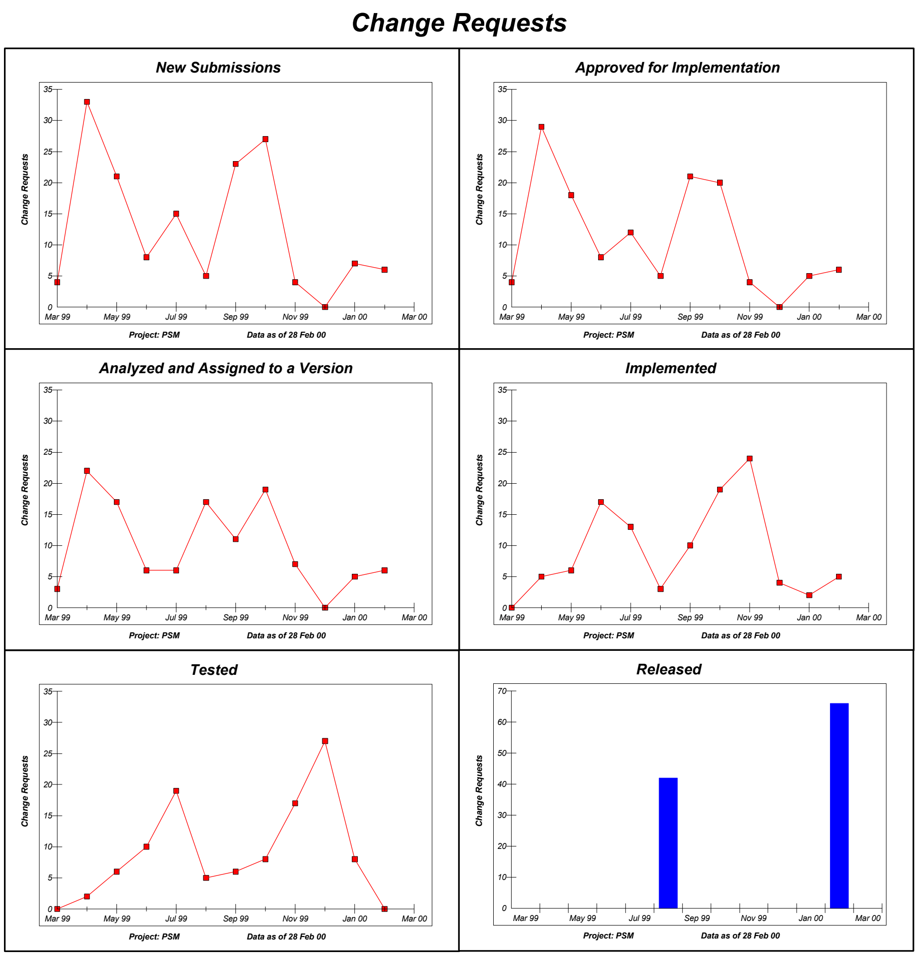

Figure 5-26a.

When assessing a system's stability relative to change requests, it helps to categorize the change requests. Use a relevant characteristic such as priority or relative impact. For example, Figure 5-26b plots change requests by priority. Analysis reveals several critical change requests early in the development life cycle, but their number, as well as the total number of change requests, has diminished as the system matured. A change request category that remains high for longer than expected indicates that the system is not maturing, or that the users are identifying new uses for the system that were not previously captured.

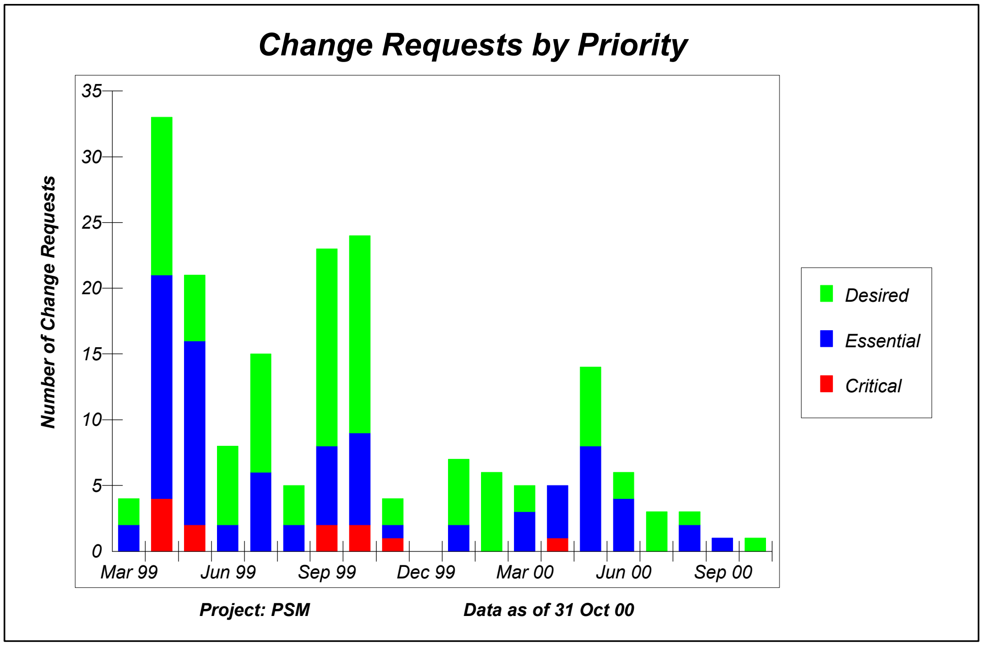

Figure 5-26b.

Additional Analysis

Change requests should be examined from several perspectives to provide a complete picture of system stability. For example, it may be useful to identify components that are affected by change requests, pointing out areas of real or perceived system weakness. Similarly, change requests could be classified as either new requirements or changes to existing requirements to assess the quality of the original requirements analysis process. Many changes to existing requirements suggest the requirements are probably not very mature.

Lessons Learned

To assess system stability using change requests, indicators must be based on characteristics of changes that can provide the greatest insight. For example, using an indicator that tracks change requests only by their potential implementation impact may overlook the fact that all requested changes were critical. In this case, the few necessary changes may have a small implementation impact. Less critical changes may not have been requested yet because the users were not able to perform any useful functions. Even though there are few changes, the software or system was very unstable. Conversely, some relatively low priority changes may substantially impact implementation.

2.16 Defects

Category: Functional Correctness

Common Issue Area: Product Quality

Applicability: Applies to most types of projects

Analysis Guidance and Examples

Defect indicators quantify the number and types of defects in a product so that quality and readiness for delivery can be assessed. Open defects represent remaining work (rework), and defect trends provide insight into product maturity. (Problem Report Status indicators provide additional information.)

Figure 5-27a plots three items describing Severity 1 (high priority) defects for this project including: 1) new defects opened within the reporting period, 2) defects closed within the reporting period, and 3) total open defects. This graph shows that the closure rate has remained relatively constant, that the number of new defects appears to be slowing, and that the result is a recent downward trend in open defects. Since the User Acceptance Test milestone is in six weeks, these indications are positive signs that the product is nearing readiness for this event. Average weekly closure and discovery rates could be used to predict whether or not the project will be ready. When predicting readiness, also analyze current and near-term testing activities, staff levels, and other scheduled events.

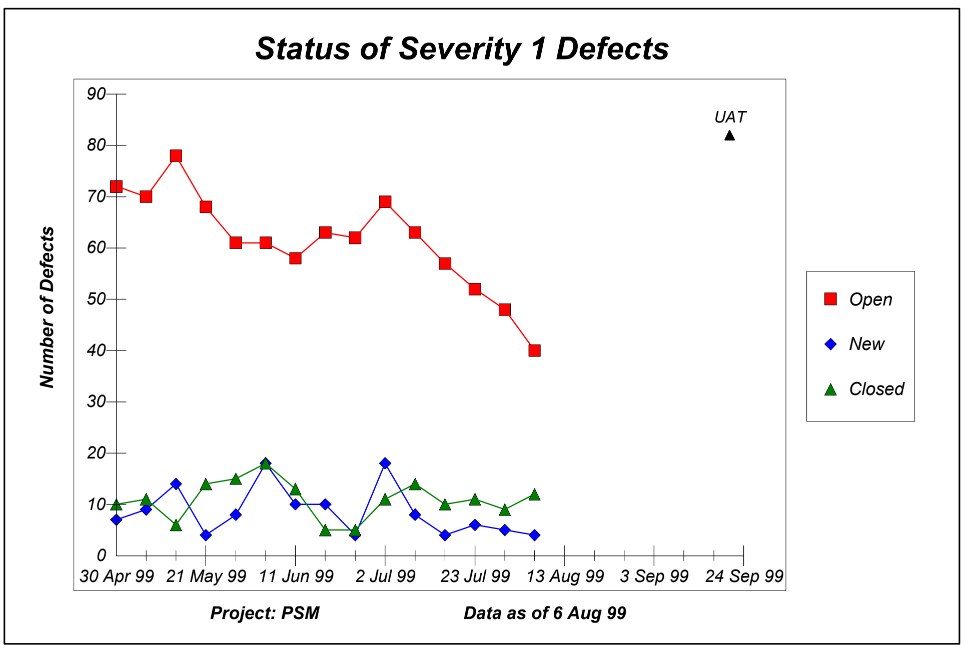

Figure 5-27a.

Figure 5-27b shows how defect density can help assess product quality throughout an iterative development process. In this process, increasing functionality is delivered to the test organization with test drops (incremental releases of functionality). In this example, defect density is the number of defects divided by lines of code. The lines of code are measured in components of 1000, expressed as KSLOC (thousands of source lines of code).

With each new test drop, functionality (and product size) is added and additional tests are run. Defects are counted for a given test drop and normalized by the lines of code delivered with that drop. The target range compares project quality to historical norms. The defect density rate for Drop B was above this organization's target range and should have been investigated. Defect density rates for all other drops were within historical ranges.

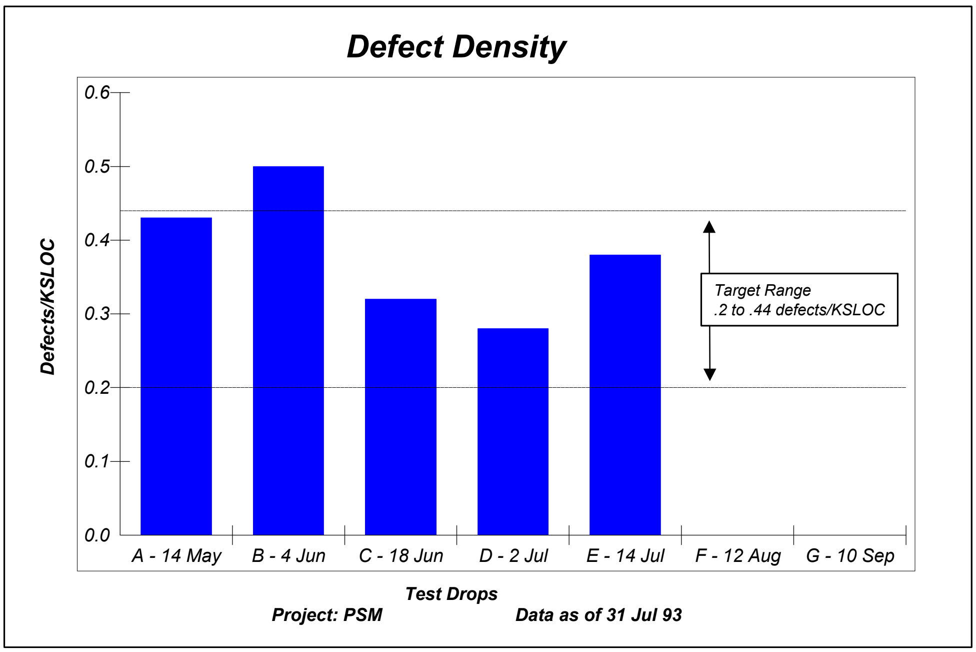

Figure 5-27b.

Use a Pareto analysis to better understand the defects and to improve product quality (by reducing or eliminating the occurrence of defects). Figure 5-27c indicates that the majority of problems identified in one component can be attributed to interface and requirements defects. The process should be improved to prevent these types of defects in the future.

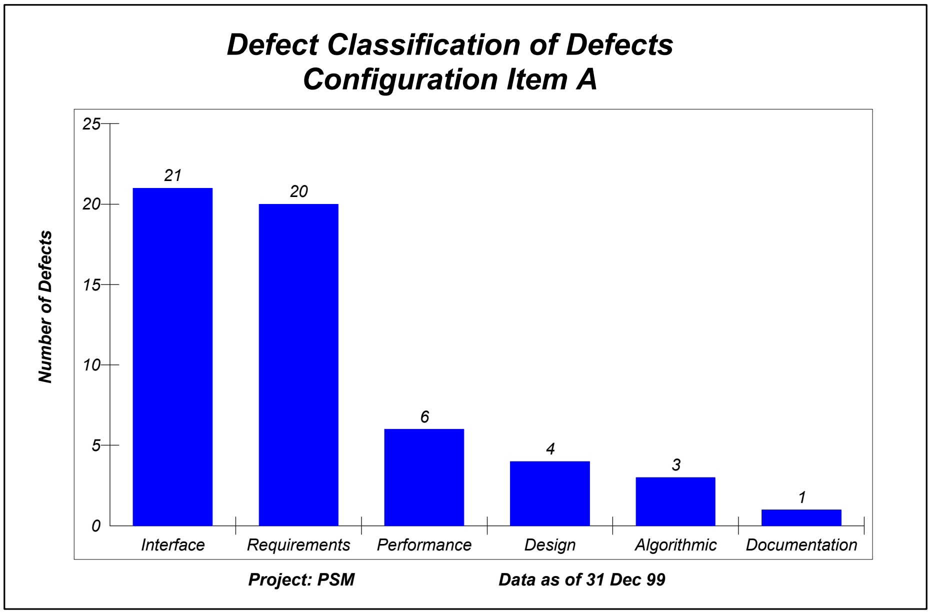

Figure 5-27c.

The final example of defect data analysis in Figure 5-27d compares historical defect densities to actual defect densities to date on a project. The project's rates are lower than expected. This could be due to process improvement and, therefore, fewer inserted defects. The lower numbers could also be due to poor defect detection early in the project, such as skipping peer reviews.

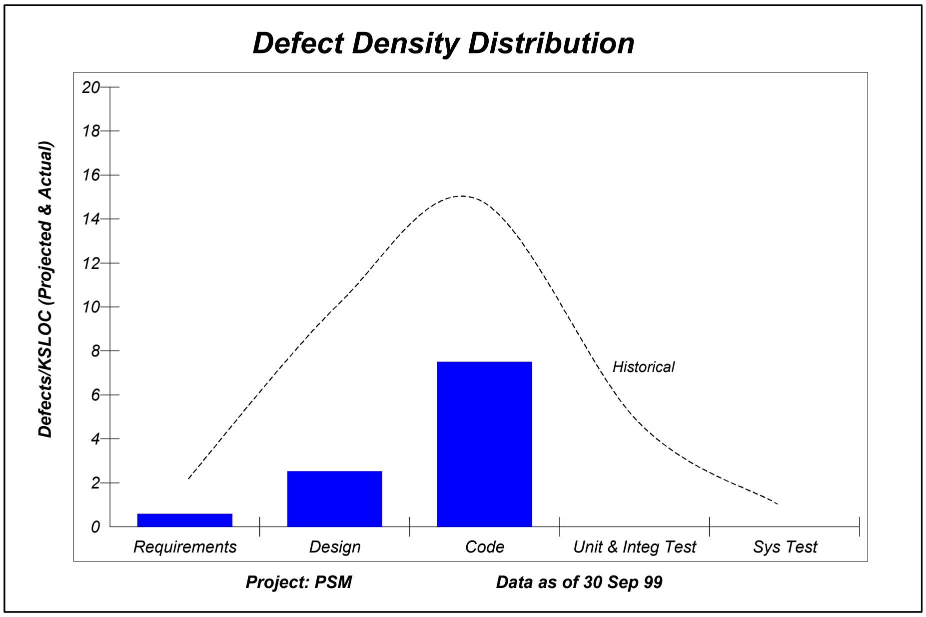

Figure 5-27d.

Additional Analysis

It is good practice to establish quantitative criteria for product releases. Typical criteria include no high-priority open defects, and a quantitative threshold on the number of remaining low-priority defects that have existing workarounds or do not pose a threat to safety.

Lessons Learned

The number of new defects found, although representative of product quality, is directly related to the thoroughness and frequency of defect discovery activities (such as audits, inspections, and tests). Product quality indicators are derived by tracking defects from design and code inspections. In general, error prone system components should be carefully inspected and analyzed throughout development.

Low defect discovery rates may be due to incorrectly performed processes, lack of preparation for reviews or inspections, or poor test scenario design.

Low defect closure rates should be considered when assessing schedule risk. Inadequate resources and rework directly affect closure rates.

2.17 Time to Restore

Category: Supportability-Maintainability

Common Issue Area: Product Quality

Applicability: Applies primarily to systems engineering

Analysis Guidance and Examples

These indicators provide insight into the time and effort required to restore the system to an operational state and to fix the cause of the failures, measured either in a test phase or during operation.

The time and effort required to restore the system may be quite different than the time and effort required to repair or fix the components that have caused the failure. Restoration may require simply switching over to a redundant component, rebooting the system, or establishing a temporary procedural work-around. Mean Time To Repair (MTTR) is an indicator that calculates the average time to fix (and verify) the causes of the failures. On the other hand, repairing or fixing the system requires identifying the cause of the failure, fixing the components causing the failure, and verifying the corrective action and operation of the system. Mean Time To Restore System (MTTRS) is an indicator that calculates the average time to restore the system to operational status. Required performance levels in terms of MTTR and MTTRS are specified for many systems.

Figure 5-28a shows a line chart that plots MTTRS and moving average MTTRS against the target of six hours. It is important to look at longer-term trends along with the shorter-term data. For example, during four periods after October 8, the MTTRS increased during one-week intervals. However, an analysis of the moving average for MTTRS indicates that the MTTRS trend is still declining. It could be unnecessary or even damaging to decide on corrective actions based on short-term data without understanding the bigger picture.

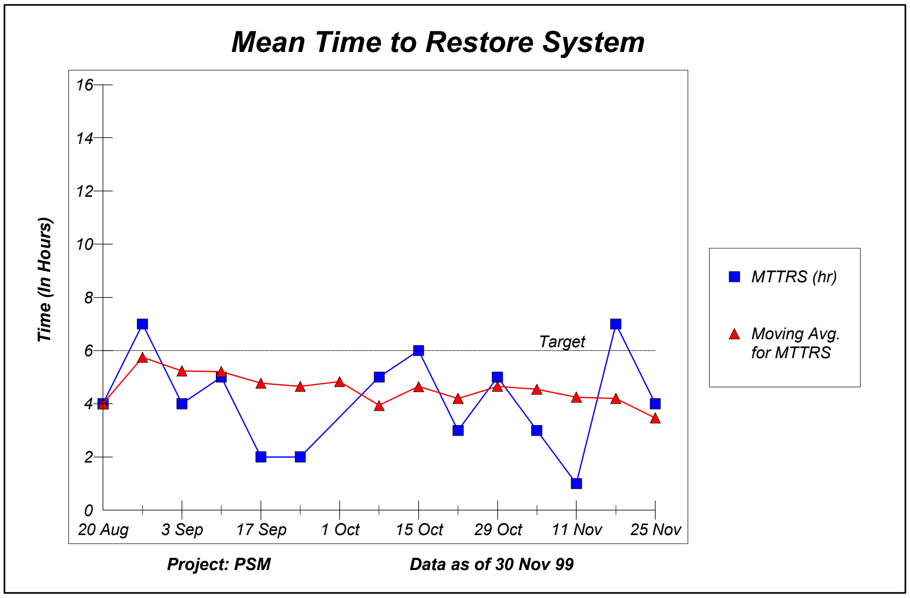

Figure 5-28a.

Figure 5-28b shows that during the first phase of testing, average repair time and effort increased but then decreased significantly. Repair time increased because of the number of failures that occurred early in the test phase. There is now a queue of required fixes that the maintenance staff must address in priority order. Thus, some lower-priority fixes are causing the MTTR to increase. This phenomenon occurred more severely during the week of October 22 and was resolved by temporarily assigning more maintenance staff to the project. At the peak of average repair effort, the majority of failures were caused by software defects that required rewritten and retested code. At the low points, many failures were caused by worn-out hardware that required replacing and inspecting components.

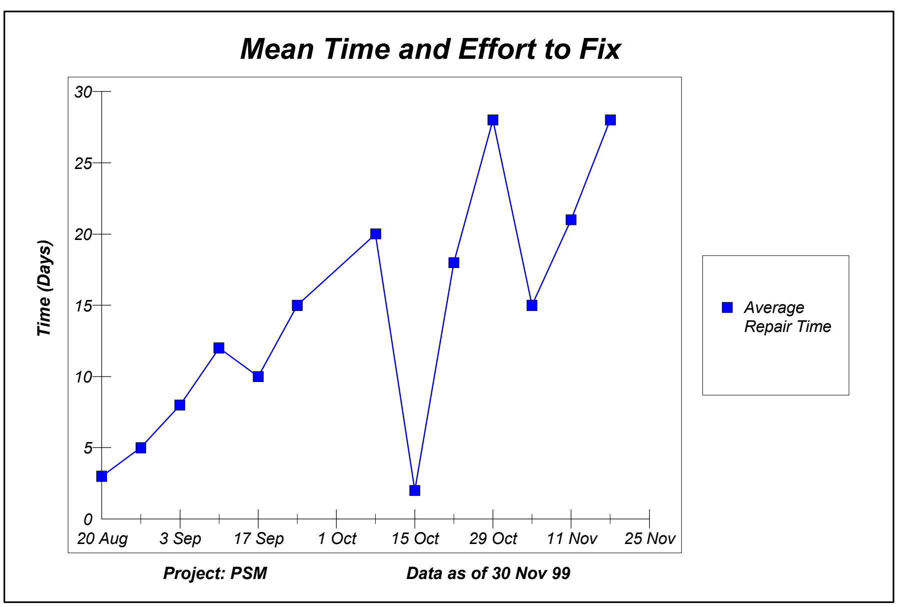

Figure 5-28b.

Additional Analysis

As testing continues, the quantity and severity of the failures tends to decrease. Review the test program plan and reports in conjunction with this data to understand these results.

Lessons Learned

These indicators help estimate the maintenance needed to support the system and the overall system downtime, providing an indication of whether system availability can be achieved. If the MTTRS information indicates significant risk in meeting the required system availability because of long recovery times, corrective actions should be considered. These corrective actions may include changing the maintenance strategies or adding fault tolerance mechanisms (such as redundancy) to the design.

2.18 Cyclomatic Complexity

Category: Supportability - Maintainability

Common Issue Area: Product Quality

Applicability: Applies to software engineering

Analysis Guidance and Examples

Cyclomatic Complexity provides insight into the difficulty of testing and understanding a software component. It measures the number of logic paths in a component to assess the amount of testing required, to predict error-prone components, to estimate future maintenance effort, or to identify components that should be considered for redesign or reimplementation. The Cyclomatic Complexity indicator compares the number of paths in unit to a standard or required threshold.

The bar chart in Figure 5-29a identifies the number of components in each complexity range. Each component within Configuration Item (CI) A was measured using an automated code complexity analysis tool. Component complexity values were separated into six complexity range categories and graphed. The threshold was also plotted.

Figure 5-29a indicates that most CI A components are less than or equal to the maximum threshold of ten for component complexity.

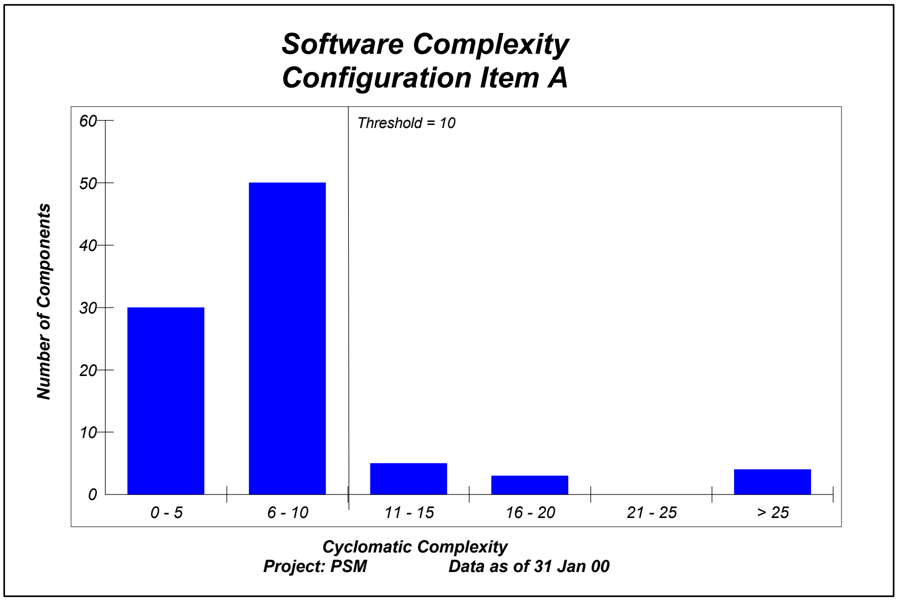

Figure 5-29a.

The table in Figure 5-29b was produced by sorting the components by complexity and showing only those components with a complexity higher than the threshold. CIs with a complexity higher than the threshold are candidates for redesign or additional review, inspection, and test.

Software Complexity
For Configuration Item A
Units with Complexity > 10

Unit	Cyclomatic Complexity
A10	53
A2	49
A12	32
A11	27
A5	20
A9	19
A7	16
A8	15
A6	15
A1	13
A4	12
A3	11
A12	11

Project: PSM   Data as of 31 Jan 98

Figure 5-29b.

Additional Analysis

It is useful to look at the defect history (if it exists) for components with high complexity. When a component is defect prone, complexity analysis helps to justify redesign, rewrites, or comments.

Lessons Learned

This measure is usually not available until after a component is coded, although it can be calculated from design specifications by analyzing flow charts or program design language. An automated code analysis tool can accurately and efficiently produce the measure from code.

2.19 Timing

Category: Efficiency

Common Issue Area: Product Quality

Applicability: Applies to both software and systems engineering

Analysis Guidance and Examples

Timing indicators compare system performance against specified performance levels. During planning and requirements analysis stages, the feasibility of meeting system-timing requirements must be assessed. The bar chart in Figure 5-30a compares actual response time data from three operational systems against a contract-specified response time requirement for online functions for a new system.

Figure 5-30a shows that the contract requirements are within the range achieved on similar projects. The requirement appears achievable, assuming that the historical systems were similar.

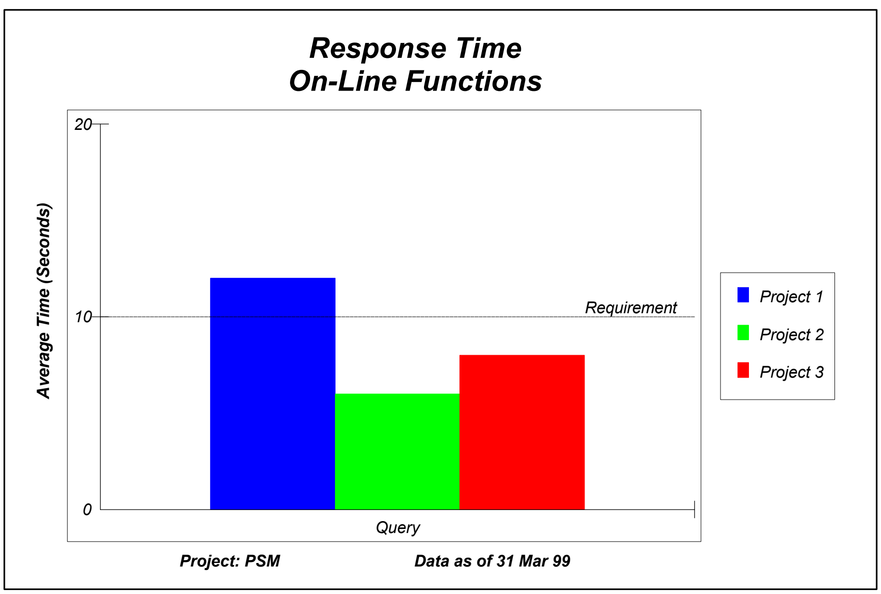

Figure 5-30a.

During project execution, response time was also monitored. A bar chart (Figure 5-30b) was used to compare the results of response time tests against a contract-specified response time requirement. A series of tests was executed for sets of representative queries and update functions. For each function, response time measures were collected with an automated tool. The collected data sets were then averaged. The graph plots three test runs and indicates the acceptable average response time as a straight line.

Figure 5-30b shows that query-type functions initially exceeded response time requirements. These functions were subsequently modified to improve performance and are now within the acceptable range. Update functions initially performed well, but performance problems were noted in the second test. These were resolved prior to the third test.

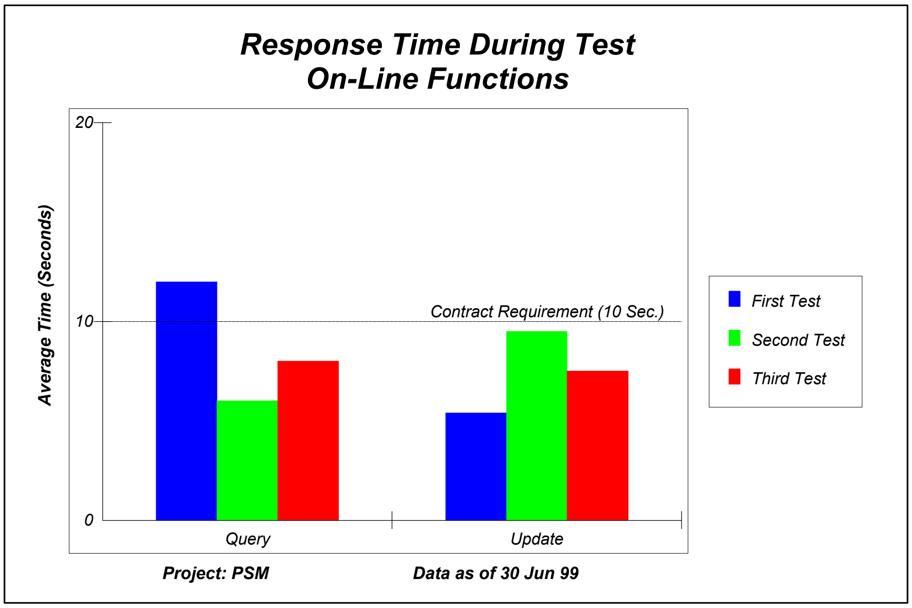

Figure 5-30b.

Additional Analysis

When comparing different systems, hardware, communications, and database technology should be similar to ensure that historical data is comparable. When results are outside the acceptable range, a more detailed analysis by component or transaction can help identify the problem.

Lessons Learned

Large decreases in response time between systems should not be expected unless technology or functionality has changed significantly. Performance for the new system can be projected with modeling and simulation techniques.

Select functions for response-time measurement based on specific criteria, such as functional similarity, criticality, or frequency of use. Also, compare the form of response-time data (an average, sample, or worst case) to the planned or target figure. Factors that may influence the validity of actual response time measures include: 1) not simulating sufficient load on the target machine during the tests, 2) not sampling representative functions, 3) not simulating a load representative of an anticipated operational profile, and 4) using a test database that is smaller than the operational version.

2.20 Standards Compliance

Category: Portability

Common Issue Area: Product Quality

Applicability: Applies to both software and systems engineering

Analysis Guidance and Examples

The Standards Compliance indicator provides insight into product interoperability for a specified environment, given that the product meets a defined set of interface requirements. The indicator compares the level of product compliance to established standard interface requirements for a given application domain (such as POSIX or ODBC). It can also assess the readiness of prototype products for the next phase of development.

The indicator in Figure 5-31 can help quantify the expected product quality or the product's readiness to proceed to the next project phase. The top line shows the total number of interface requirements to be validated for the product. A second line shows the planned validation path for checking the interfaces. A third line represents the cumulative number of requirements successfully tested each week.

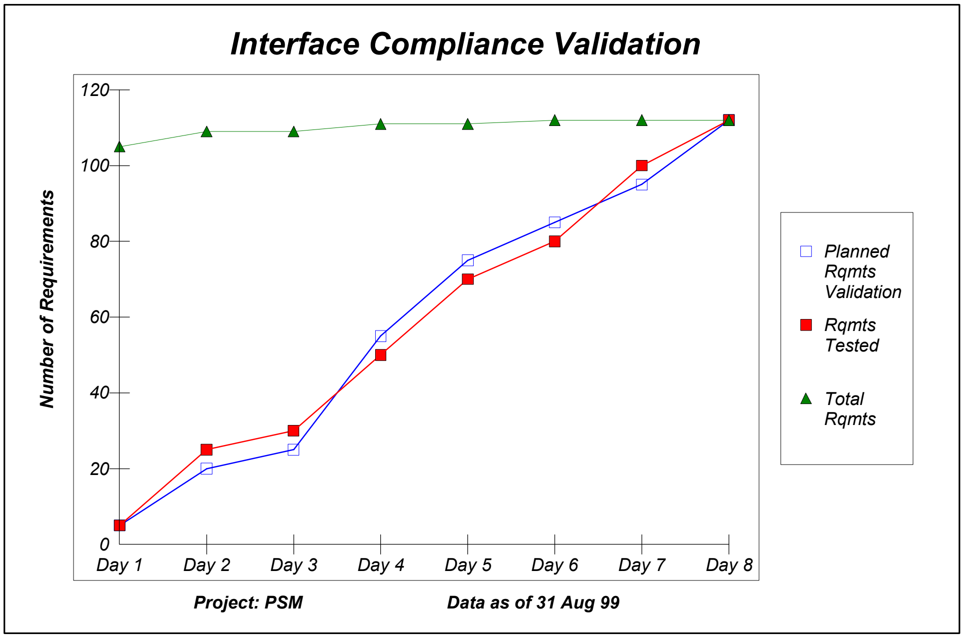

Figure 5-31.

Additional Analysis

For this product to be used in other operating environments, additional requirements may be needed. Analysis of all prospective operating environments should be conducted and classified by affected functional area (such as system, operator, digital, or analog) and mapped to validation methods and testing levels.

Some requirements may be validated by means other than testing within the target environment. Many requirements associated with multiple environments may be validated with the same or similar tests. Validation of interface requirements can be completed through mathematical analysis, product inspections, component test, system test, or integration test.

Additional interface validation requirements can be added to the plan after the project has begun. On the other hand, some requirements may be removed if design or implementation decisions render them unnecessary.

Lessons Learned

Validation of interface requirements lends itself to portability among homogenous environments, but additional types of validation are needed when porting across dissimilar environments.

2.21 Operator Errors

Category: Usability

Common Issue Area: Product Quality

Applicability: Applies to both software and systems engineering

Analysis Guidance and Examples

The Operator Errors indicator provides insight into system failures or anomalies that are attributed to operational causes rather than to hardware or software discrepancies. Analysis of these indicators identifies where system ease-of-use or knowledge-of-use could be improved. Examination of the errors' causes often identifies corrective actions to reduce or eliminate the errors. For example, a recurring operator error may be caused by an error in an operations manual, difficulty reading a display, or inadequate training, all of which can be fixed after the problem is identified.

Figure 5-32a shows the results of analyzing "type of problem" data collected in system problem reports for the past six months. Operator errors account for 26% of the current problems. Since this was such a significant portion of the problems reported, an analysis was performed to find the underlying cause.

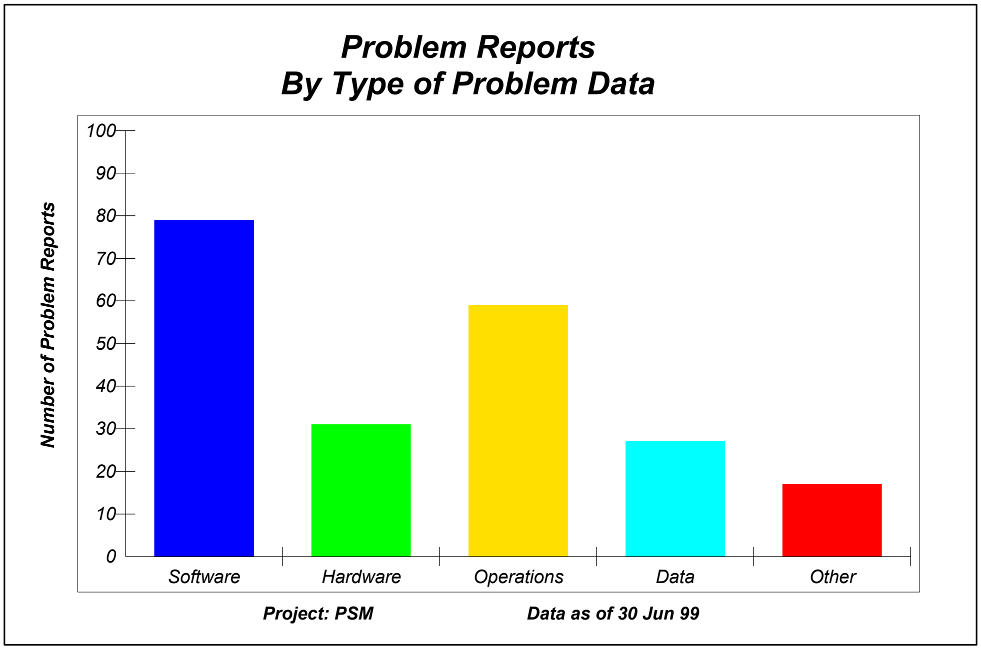

Figure 5-32a.

Figure 5-32b graphs the reasons for operator errors in Pareto format (greatest to least number of problems).

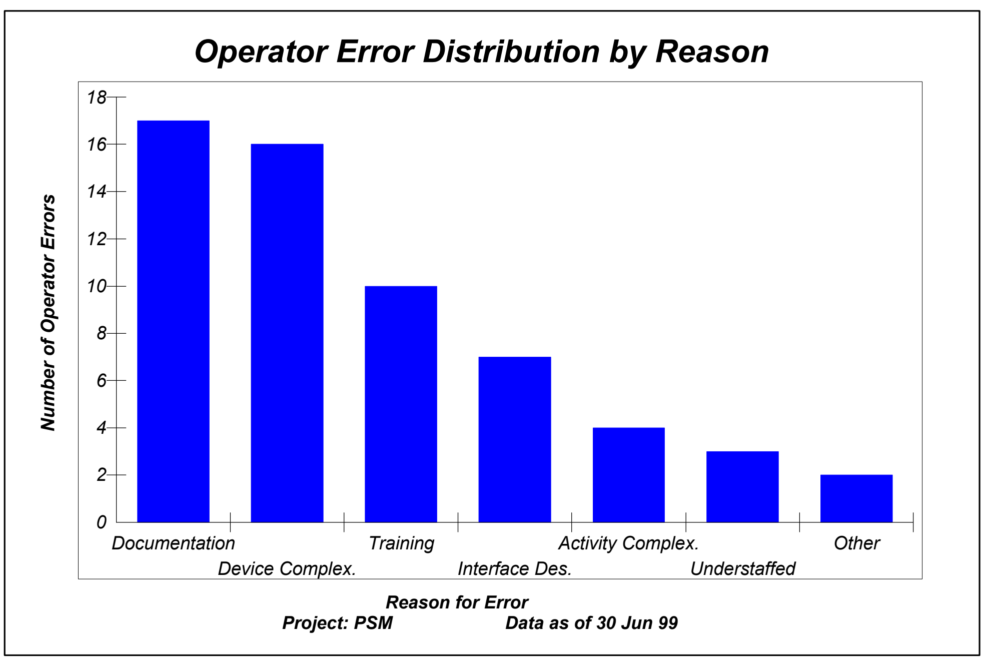

Figure 5-32b.

The distribution shows that documentation and device complexity are the major causes of operator errors. They should be analyzed further to determine if a specific type of device is too complex or a specific function is not being adequately taught. For example, Figure 5-32c categorizes the user interface devices by the "device complexity" data item. The complexity of the control panel is clearly a major cause of operator errors.

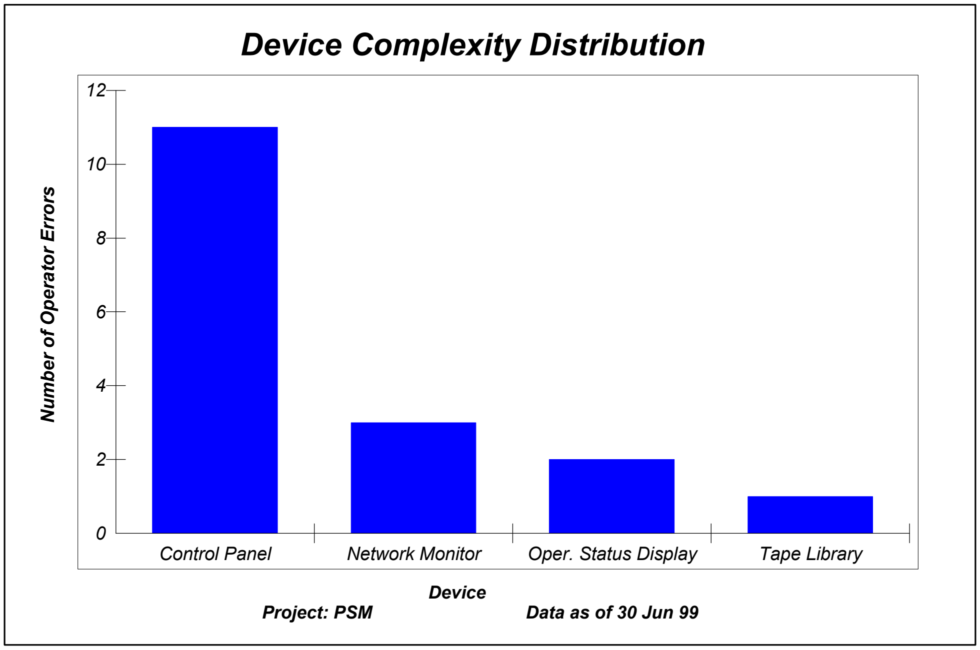

Figure 5-32c.

Additional Analysis

Collecting data on other attributes (such as priority, severity, or timing of operator errors), can help determine other corrective actions. For example, error timing may correspond to the installation of new equipment or to changes in the operational procedures. It is also useful to analyze the distribution of the operator errors over time.

Lessons Learned

Use ranked distributions of the data (such as Pareto charts) to identify the greatest opportunities for improvement.

2.22 Failures

Category: Dependability - Reliability

Common Issue Area: Product Quality

Applicability: Applies to both software and systems engineering

Analysis Guidance and Examples

The Mean Time Between Failures (MTBF) indicator is often used to provide insight into system or software failure trends. This indicator shows the average time from one failure to the next, in operations or test. MTBF is often a system performance requirement, tracked as a technical performance measure. After the desired MTBF requirement is established, it should be checked for feasibility against the system or software application and tracked to monitor performance against plan.

During planning and requirements analysis, assess the feasibility of meeting stated reliability requirements. Compare reliability requirements against historical performance data from similar systems. If the requirements are too stringent for the system type, it may be difficult to achieve the required MTBF, or it may not be a cost-effective design. However, if the requirement is lower than the range historically achieved, then operational performance may be in jeopardy. Figure 5-33a graphs ranges of historical data for each system application, to help build reliability plans and to perform an MTBF feasibility analysis.

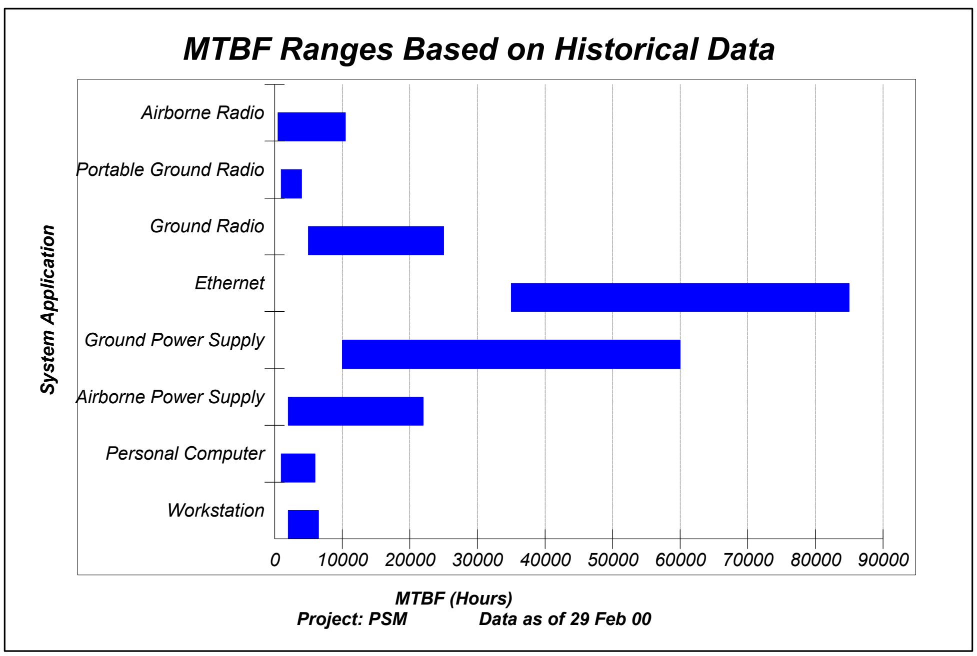

Figure 5-33a.

After choosing the target MTBF, compute intermediate target values and milestones for MTBF achievement. Once the project is underway, tracking MTBF involves collecting the time intervals between consecutive failures and deriving the arithmetic average. As testing activities identify failures in components and performance, the root causes (such as requirements, hardware design, or software implementation) are addressed, and the MTBF generally grows toward the required value. The evolution of the MTBF, called reliability growth, is used to determine if reliability is improving at a rate necessary for the system to be fielded on schedule and to be used successfully.

Figure 5-33b is an example of a reliability growth plan and the tracking of actual MTBF performance against the plan. In this example, the project was performing well against the plan during the first two months. However, the trend changed in March as reliability growth fell below the target value. A single month's performance was not enough to initiate major actions beyond understanding the causes of the change. When the variance from plan increased in April, the project identified the root causes of the problem and took corrective actions to improve performance. This intermediate tracking of MTBF helped to identify the growing risk of not meeting the required MTBF and to determine whether the corrective actions were working.

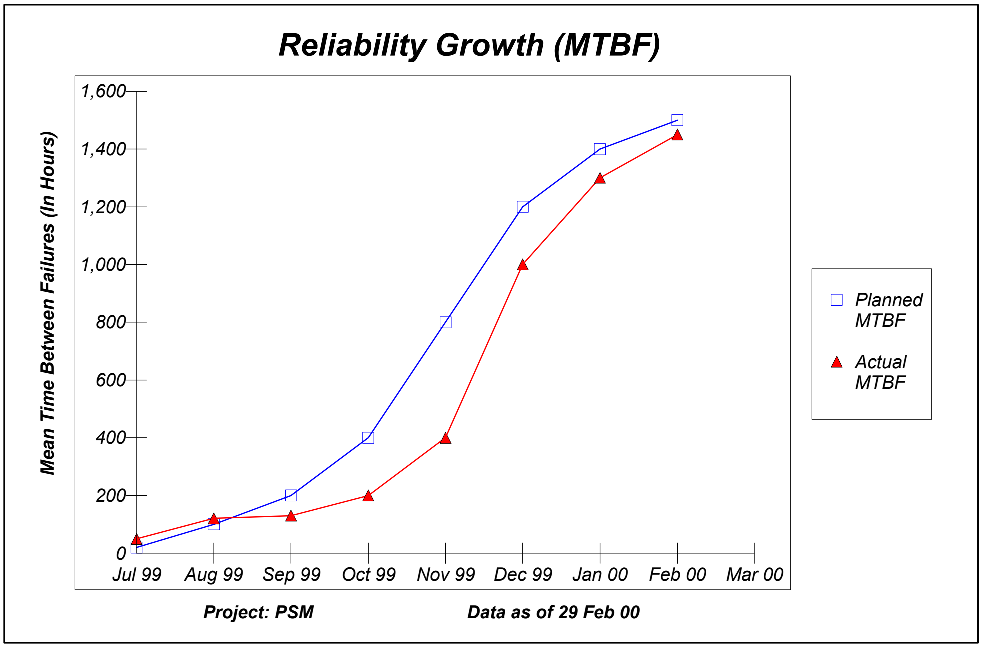

Figure 5-33b.

Additional Analysis

If enough data is maintained to assess intermediate MTBF performance, the organization can establish reliability growth profiles for various types of system and software applications. Since these profiles are based on historical data, they would be representative of the organization's capability for reliability growth. Establishing profiles would not only improve the accuracy and efficiency of planning (estimation and feasibility analysis), but it would also provide a mechanism for identifying and measuring process improvement opportunities.

Lessons Learned

Use MTBF information as one of the quality gates for releasing a system into the operational environment.

The system-level MTBF should be calculated after integration testing (using projected operational profiles) to obtain realistic estimates of the fielded system's dependability.

2.23 Reference Model Rating

Category: Process Compliance

Common Issue Area: Process Performance

Applicability: Applies to both software and systems engineering

Analysis Guidance and Examples

The Reference Model Rating indicator provides insight into an organization's systems or software development capability. It assesses the organization's engineering, project management, and organizational support processes relative to a standard model of good engineering practices. This measure can help to evaluate an organization's process improvement effectiveness, to determine an organization's current process performance, or to select among competing bidders.

Use the indicator to answer questions relating to a process improvement program:

- Is the project or organization meeting its process performance goals?
- Is the project performing at the same level of performance as the rest of the organization?
- Is the process improvement program working?

Commonly used reference model include the Systems Engineering Capability Maturity Model (EIA/IS 731), Capability Maturity Model® Integrated Systems/Software Engineering, and ISO/IEC 15504 (Process Assessment). Reference models may be continuous or discrete. Figure 5-34a shows an indicator for a continuous-type reference model, where each process area (or focus area) is evaluated independently. With this indicator, an organization can evaluate ongoing progress of its improvement efforts for each selected process area or for overall performance. For process area 1, the organization has made steady progress. For process area 2, process capability decreased between the first and second assessment.

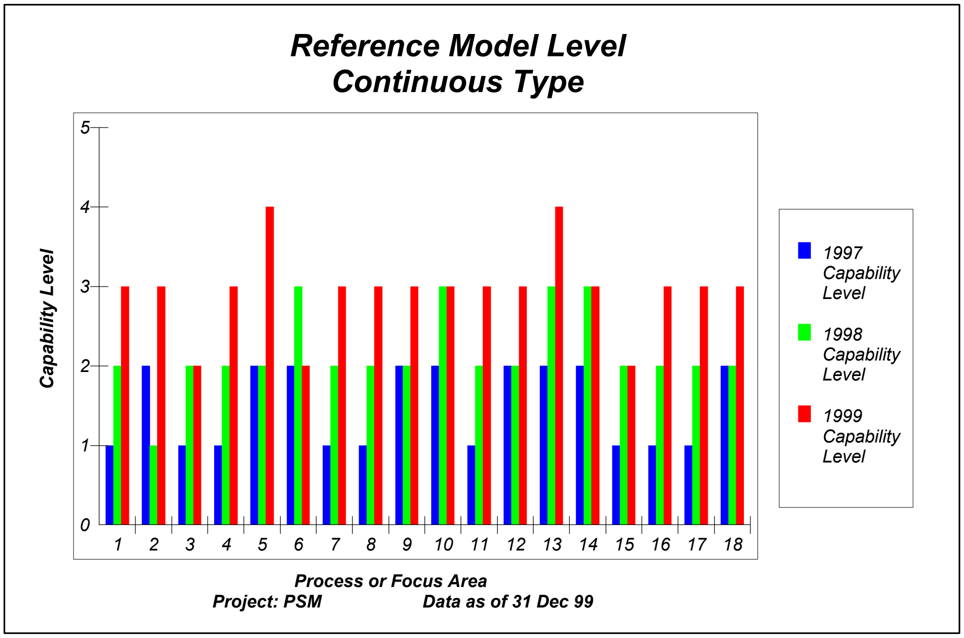

Figure 5-34a.

Figure 5-34b shows an indicator for a staged reference model, such as the Software Capability Maturity Model (SW-CMM) or the People Capability Maturity Model. In a staged model, the maturity level is achieved by successfully implementing key process areas, yielding a single rating for the organization. This indicator shows the organization's overall process improvement progress over time.

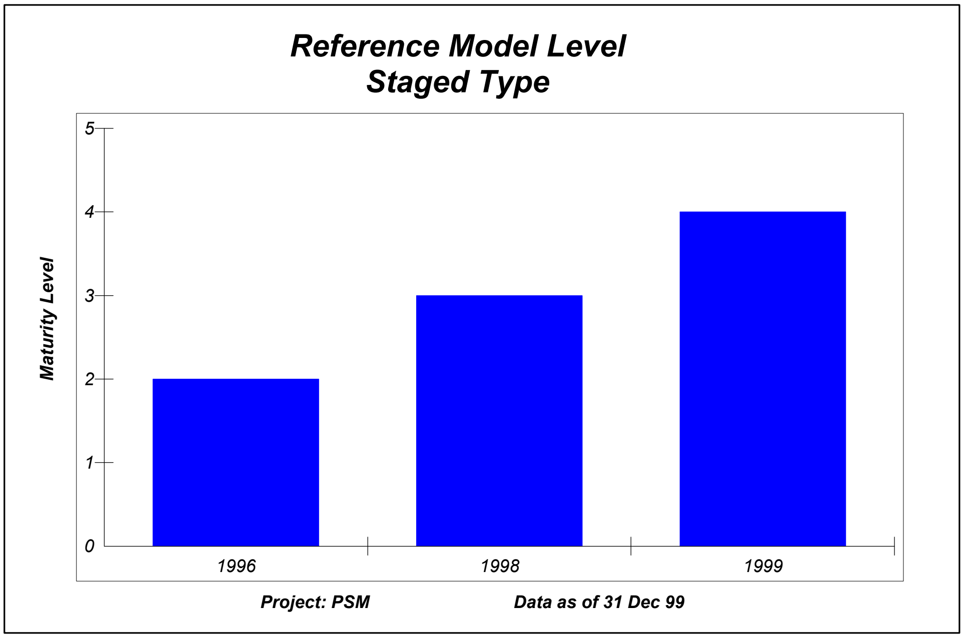

Figure 5-34b.

Additional Analysis

Overall reference model ratings (such as Figures 5-34 a and b) provide little insight for managing process improvement. Maintaining a detailed list of model practices addressed provides a better indicator of progress.

Lessons Learned

The reliability of the process maturity score depends on the rigor of the assessment process. A high-maturity score does not guarantee successful development. Project constraints can significantly influence an organization's ability to implement the defined process.

It is important to periodically audit process compliance of all processes to maintain process capability even if the assessed level is satisfactory.

2.24 Process Audit Findings

Category: Process Compliance

Common Issue Area: Process Performance

Applicability: Applies to both software and systems engineering

Analysis Guidance and Examples

This indicator monitors a project or organization's adherence to their defined processes. Use it to determine how well the project or organization is conforming to standards or processes. Process Audit Findings indicators track results from periodic process audits that are conducted to ensure process quality and compliance.

Figure 5-35a is a bar chart of findings from project process audits over a three-year period. The results show process improvement elements where compliance is problematic. Processes 1 and 4 have shown significant and steady improvement, while processes 8 and 9 continue to need improvement. The ongoing findings for these processes could show ineffective corrective actions or new problems related to implementation difficulties. The lower level details of these findings, such as shown in Figure 5-35b, can help identify causes.

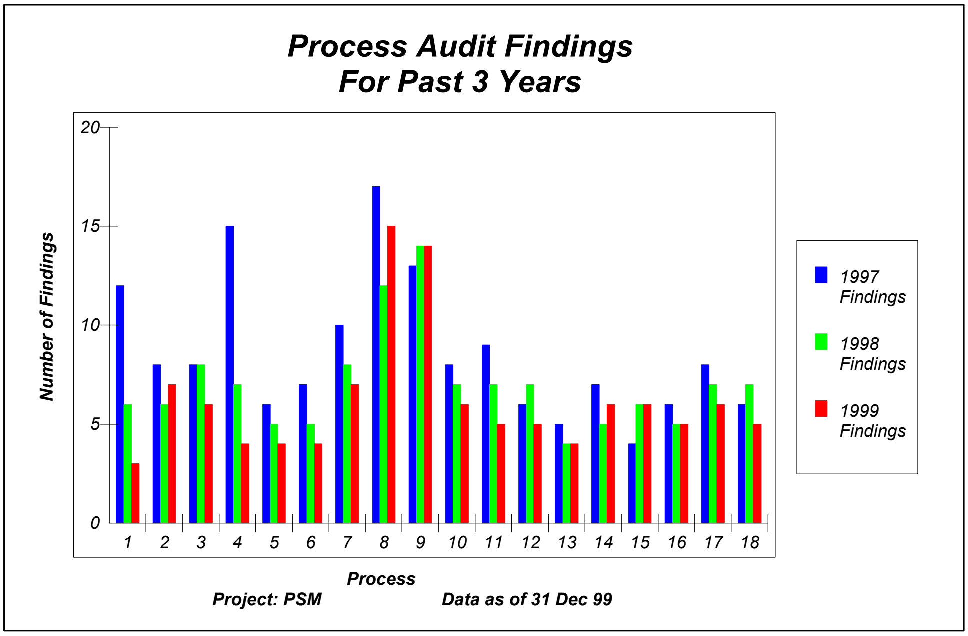

Figure 5-35a.

Figure 5-35b breaks down the current year's findings by reason code, to determine if there is a common cause for the problems. It appears that inadequate process training (reason code 3) is the main cause of the process findings. This insight should lead to an enhanced training program for these (and possibly all) processes.

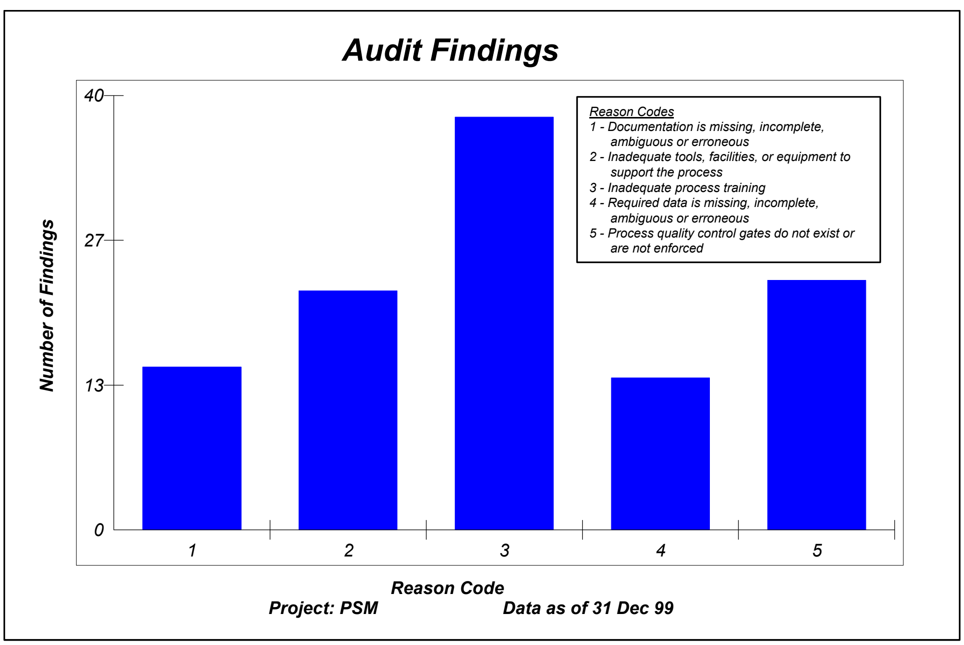

Figure 5-35b.

Additional Analysis

Tracking the time required to resolve findings helps to ensure that problems are corrected in a timely manner.

Looking at results by project helps to identify projects that are not performing in compliance with established processes. This information can help the organization determine where to focus limited resources on corrective actions.

Lessons Learned

Collect and plot data for reasons as well as for the number of findings. This data is valuable in determining what to improve and whether to initiate the improvement effort at the project or organizational level.

2.25 Productivity

Category: Process Efficiency

Common Issue Area: Development Performance

Applicability: Applies to both software and systems engineering

Analysis Guidance and Examples

The productivity indicator allows the comparison of work rate efficiency.

During planning and contracting for a project, productivity can be analyzed to evaluate two alternative bidders on a contract. Productivity can also be used by the bidder to establish estimates or to check the feasibility of an estimate.

Figure 5-36a shows data from three similar historical software projects for each bidder, along with their proposed productivity estimate. Each bar was produced by dividing the product size measure of Source Lines of Code (SLOC) by the measure of software work effort in staff months. Project effort included software requirements analysis, design, implementation, and integration and test activities.

In Figure 5-36a, the successful bidder appears to have a reasonable estimate. This bidder’s historical productivity data was in a relatively small range, and the proposal required only a modest increase in this rate. In contrast, the unsuccessful bidder had widely varying historical data. In addition, the proposed productivity was significantly higher than the productivity achieved on the historical projects, making the estimate appear unrealistic. (Planned or proposed productivity rates should always be compared to past projects with similar characteristics in areas such as tools, methods, staff skills, and programming language.)

If the unsuccessful bidder had performed a feasibility analysis of its proposed productivity rate against its historical rates, the bidder might have detected that its proposal was unrealistic.

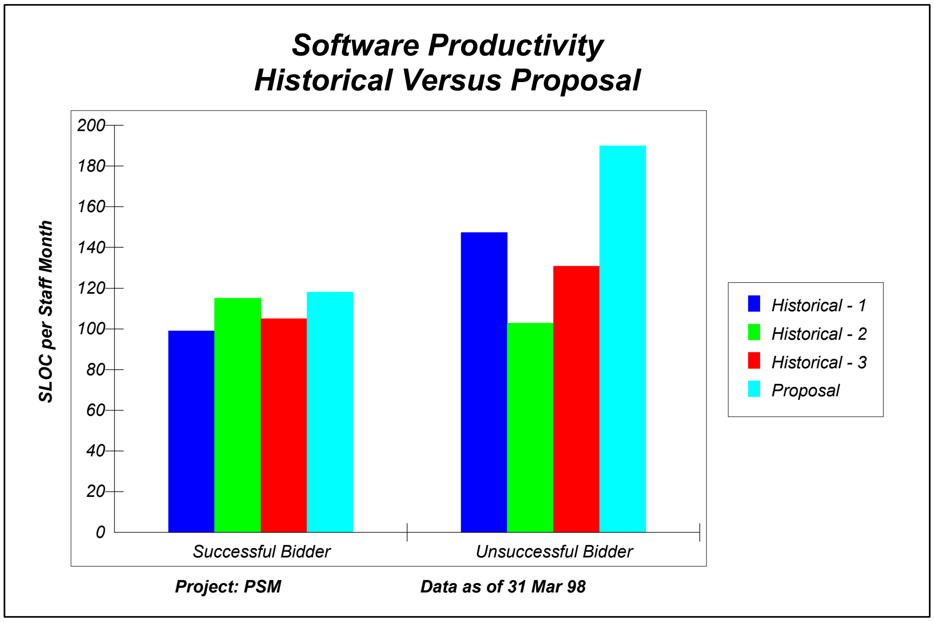

Figure 5-36a.

A second example (Figure 5-36b) shows how productivity was used to evaluate replanning options during the course of a project. The project's planned productivity rates were compared to the current actual rate and to the proposed replan rates.

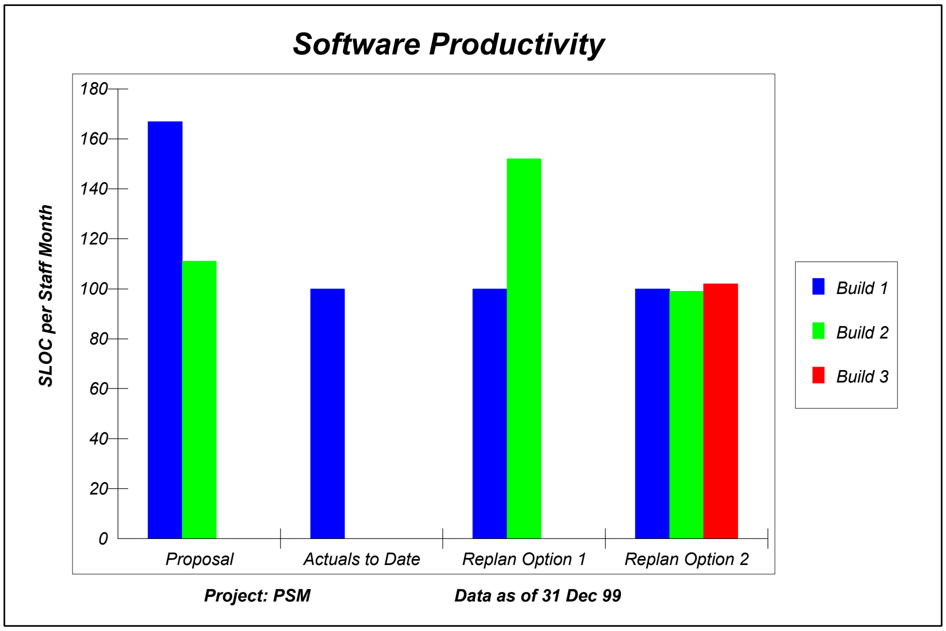

Figure 5-36b.

Figure 5-36b shows that actual productivity on Build 1 is significantly lower than planned. About 170 SLOC per staff month was planned for Build 1, and 110 was planned for Build 2. However, with Build 1 well under way, actual productivity is only 100 SLOC per staff month. Productivity must be increased, or substantially more time and effort will be needed to complete the product.

Two alternative replans were considered. Option 1 proposed slightly increasing productivity by the end of Build 1 and substantially increasing it for Build 2. Option 2 assumed the productivity rate would be similar throughout the remainder of the project to what had already been achieved, and a new Build 3 was added to complete production. Option 2 appears to be a more realistic alternative unless major changes are made in the process.

Additional Analysis

Further analysis should determine the cause of lower-than-expected productivity, before deciding on a course of corrective action. Consider issues such as learning curve, requirements volatility, and expected staff turnover when evaluating the feasibility of a proposed productivity rate. The underlying reasons for significant changes in productivity rates should be determined. Unplanned rework is a frequent cause of low productivity.

Lessons Learned

Once established, it is hard to change productivity on an existing project.

2.26 Defect Containment

Category: Process Effectiveness

Common Issue Area: Process Performance

Applicability: Applies to both software and systems engineering

Analysis Guidance and Examples

When a defect is discovered, a determination of where the defect was inserted should be made. Usually this is described as the originating phase. The Defect Containment measure helps determine how many defects inserted in a particular phase were undiscovered in that phase (that is, they escaped detection until a later phase of the project). Late defects indicate ineffectiveness in the defect discovery process during a phase.

Figure 5-37 is an example of an indicator showing requirements defect containment. It shows how many requirements defects were undiscovered during the requirements phase, only to be discovered in subsequent phases. The project in Figure 5-37 has not been delivered to the customer, so there are no entries for the operations period.

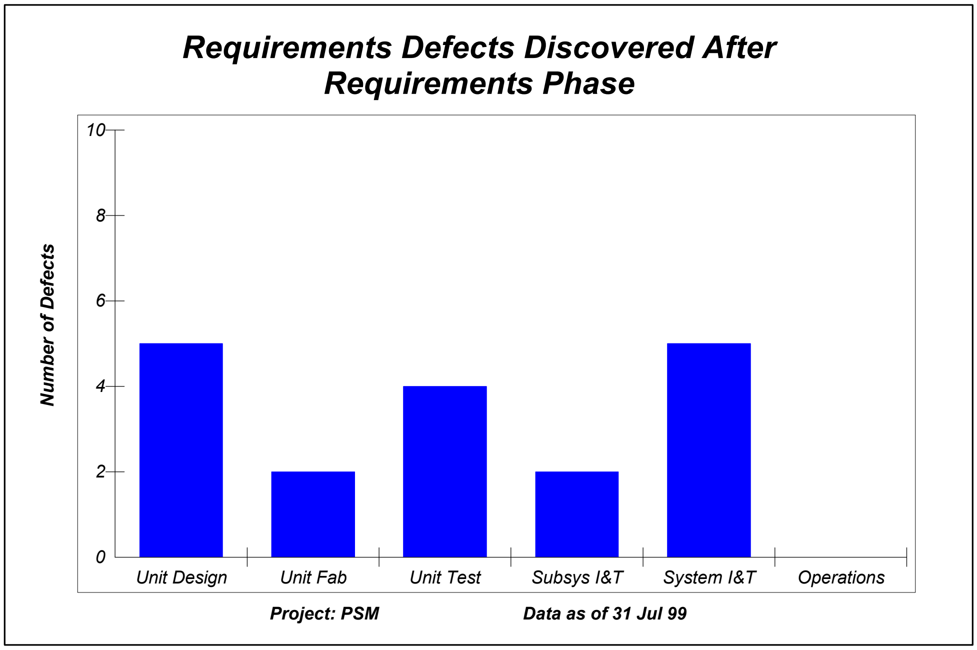

Figure 5-37.

Additional Analysis

Problems resulting from poor requirements are more costly when they are discovered later in the project, resulting in schedule delays and cost overruns. The rework effort to fix requirements defects found later in the life cycle should be evaluated, as well as its impact on schedules and budgets.

Improving the requirements definition process or the requirements inspection process will reduce requirements defects discovered late in the development. Further analysis into which components of the system have the majority of late requirements defects and analysis of the specific types of late requirements defects would help to focus improvement efforts.

Lessons Learned

The results of the defect containment analysis and associated cost information can be used to justify investments in process improvement activities.

2.27 Rework

Category: Process Effectiveness

Common Issue Area: Process Performance

Applicability: Applies to most types of projects.

Analysis Guidance and Examples

The Rework indicator compares the amount of effort being expended to fix defects to the budget for that activity. It can provide insight into the reasons for schedule slips and quality problems.

During the planning stage, rework should be accounted for in project schedules and budgets. Often, rework is estimated as a percentage of overall effort and then distributed across project phases. These percentages and distributions are derived from reviews of actual rework figures in past projects.

Two in-progress rework indicator examples are shown below. The first example (Figure 5-38a) reports rework as a category of work effort separate from development work. This indicator tracks planned and actual effort in each life-cycle phase, but cumulates all rework effort. The indicator shows that rework prior to integration and test has already exceeded the total plan by over 100 percent. However, this chart does not identify the activity in which the rework occurred.

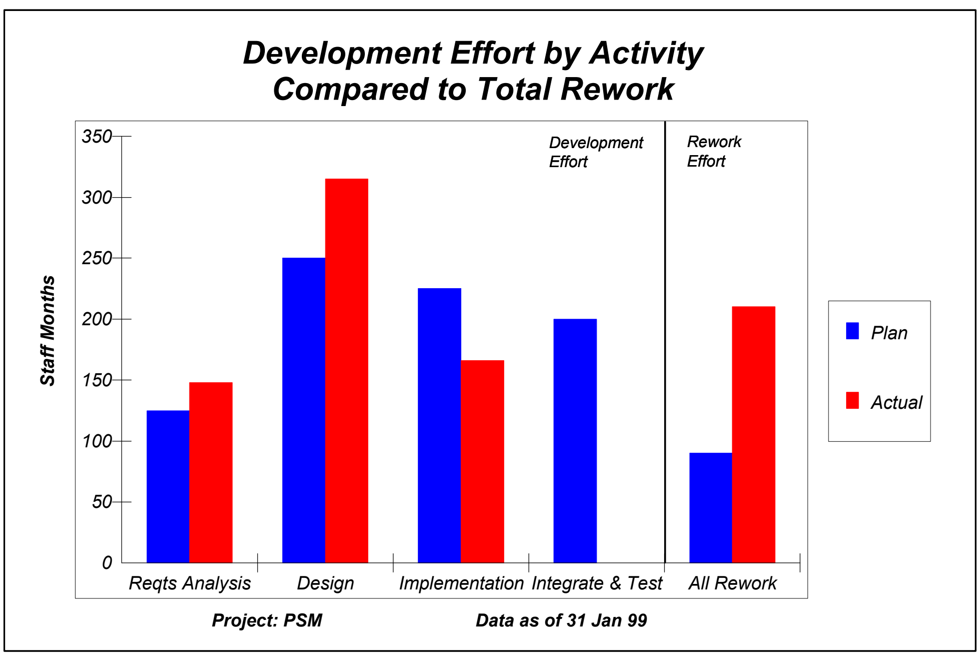

Figure 5-38a.

The second example (Figure 5-38b) was produced by an organization whose time reporting system supports the collection of rework data at the major activity level (such as requirements analysis, design, implementation, integration and test). For each chart, the accumulated number of planned and actual hours is plotted. This example shows that rework during both requirements analysis and design was much greater than expected.

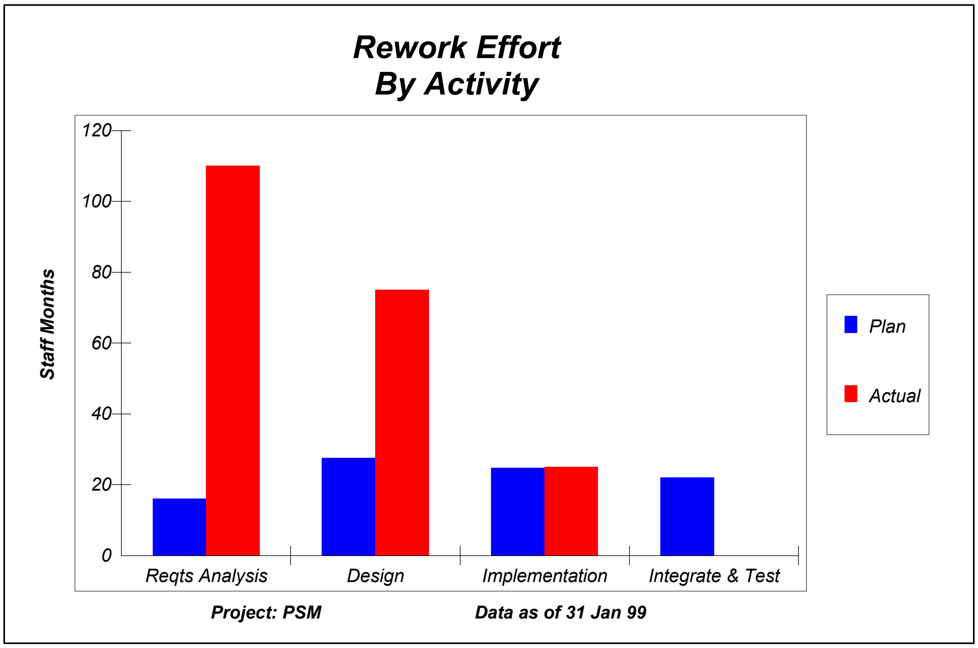

Figure 5-38b.

Additional Analysis

Taking extra time to do things right the first time can reduce overall rework. Achieving lower amounts of rework typically requires early defect detection techniques, such as reviews, inspections, and higher levels of process capability. Monitoring the performance of those activities helps control rework.

Lessons Learned

Some organizations have problems planning for or tracking rework because their time-accounting systems do not track rework as a separate task. In these cases, rework may be monitored using reviews, inspections, or problem report data. No matter how the data is gathered, rework must be clearly defined.

2.28 Requirements Coverage

Category: Technology Suitability

Common Issue Area: Technology Effectiveness

Applicability: Applies to both software and systems engineering

Analysis Guidance and Examples

The Requirements Coverage indicator helps evaluate the amount of functionality that will be addressed by a specific technology, such as a software package, hardware, or other off-the-shelf component.

Figure 5-39a shows a simple chart that combines both data and an indicator. It lists the critical requirements for a network cable tester that will be part of a complete network system. Each evaluated cable test product appears in a separate column. A cell is colored or shaded when that product satisfies the requirement listed in the corresponding row. A cell with no coloring or shading indicates that the product does not meet the corresponding requirement. Cells that are not included in the scoring, such as those used for requirements headings, are distinguished by a different color, shading, or pattern. The scoring of each product against the requirements is the data. The scores for each product appear at the bottom of the chart and are calculated by summing the number of requirements satisfied and computing the percentage of requirements covered. This is the indicator that helps choose among the products.

Critical Requirement	***Cable Tester A***	***Cable Tester B***	***Cable Tester C***	***Cable Tester D***
CABLE TYPES SUPPORTED				
Type 1				
Type 3				
Category 3				
Category 5				
Thin coax				
Thick coax				
Multimode fiber				
Single-mode fiber				
TYPES OF CONNECTORS				
RJ-45				
ST				
SC				
TESTS AVAILABLE				
Shorts				
Opens				
Wire map				
Reversed polarity				
Split pairs				
NEXT				
Signal-to-noise ratio				
Capacitance				
Attenuation				
Attentuation-to-crosstalk ratio				
Testing frequencies > 99 MHz				
Weight < 16oz.				
Battery-powered				
# Requirements satisfied	21	12	16	10
% Critical requirements satisfied	88%	50%	67%	42%

Figure 5-39a. Critical Technology Requirements

As a selected set of technology is incorporated into a project, "fit" should be reassessed. Figure 5-39b shows how ongoing assessments can highlight the impact of shifting assumptions regarding the use of purchased or existing technology. For example, in the case of network cable testers, the earlier evaluation of technology coverage may be overridden by a decision to use wireless local area network technology. This would eliminate the requirement for cable testers, but would raise new requirements for (and possible development of) specialized test equipment.

Figure 5-39b indicates that the project originally expected to meet more than 50 percent of its requirements with COTS components. However, by the detailed design phase, the expectation had shrunk to less than 10 percent. Additionally, the overall number of requirements had increased substantially. These shifts will impact most major planning assumptions such as staffing requirements, schedules, and budgets.

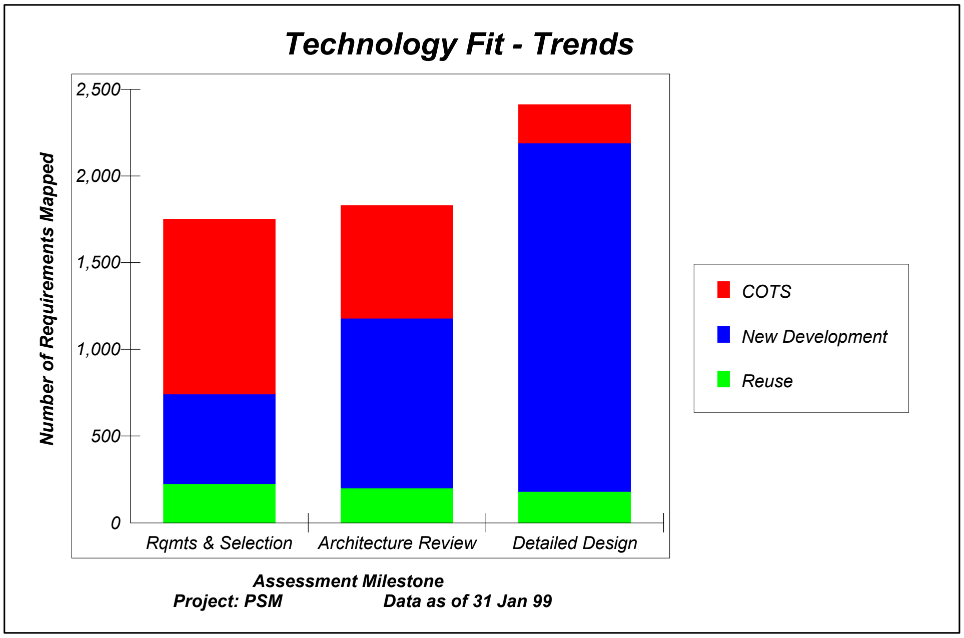

Figure 5-39b. Technology Fit - Trends

Additional Analysis

Requirements reallocated from COTS packages to developmental code or hardware must be analyzed to determine effort, cost, and schedule impacts.

Lessons Learned

Simple (re) allocation of requirements to COTS packages does not address the time and effort needed to integrate them into the system. This effort can include customizing interfaces, writing "glueware" or patch code, and performing integration tests. These items must be estimated, planned, and tracked separately.

2.29 Technology Impact

Category: Impact

Common Issue Area: Technology Effectiveness

Applicability: Applies to both software and systems engineering

Analysis Guidance and Examples

The Technology Impact indicator assesses the impact of a specific technology on a project or system. It compares competing technologies or quantifies the impact of a single technology. The assessment may be in terms of cost, performance, schedule, risk, or other factors. An example using several different criteria is illustrated below. The system under development is a medical x-ray machine that employs advanced image processing. Three candidate image-processing approaches are being analyzed. The user must decide which technology to employ, based on both cost and performance. The machine must develop an exposed x-ray picture in under 35 seconds, using the new image processing capability and at a cost of under $50 per print.

Each candidate image processor was analyzed for speed, cost per print, and estimated yearly operational cost. Sample runs of 100 exposures were made using each candidate machine. The average processing time was calculated for each. Figure 5-40a plots the three means, with a 95% confidence limit for each technology.

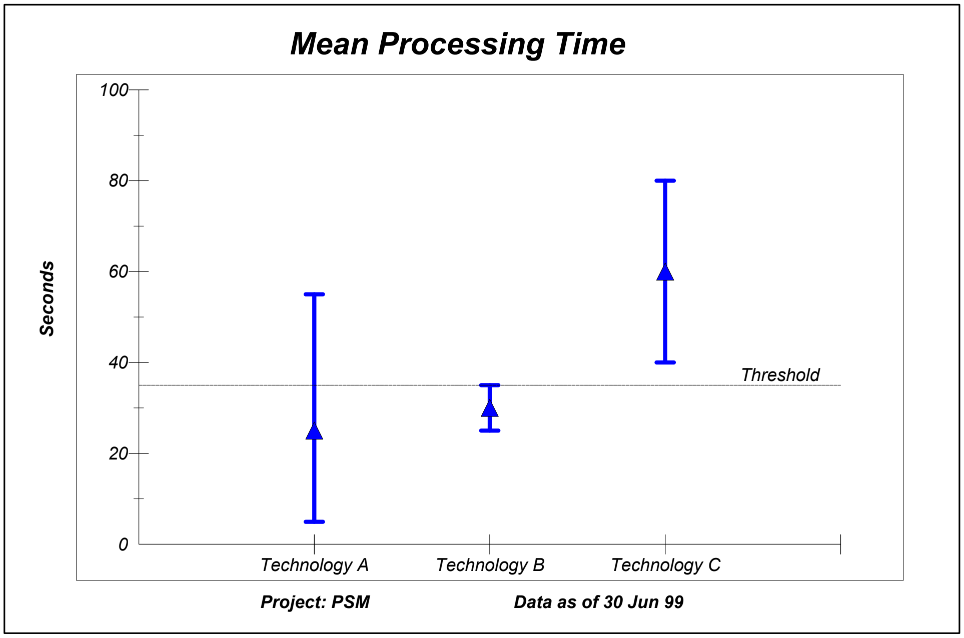

Figure 5-40a.

Figure 5-40a indicates that, based on processing time alone, technology A is better than the others and easily meets the requirement of 35 seconds. However candidate B is close and has less variable results than A (evidenced by the smaller confidence limits). The candidate using technology C is clearly unacceptable from a performance standpoint.

However, looking at cost performance in Figure 5-40b, technology A exceeds the maximum allowable cost (e.g., threshold), while B and C are both under the threshold.

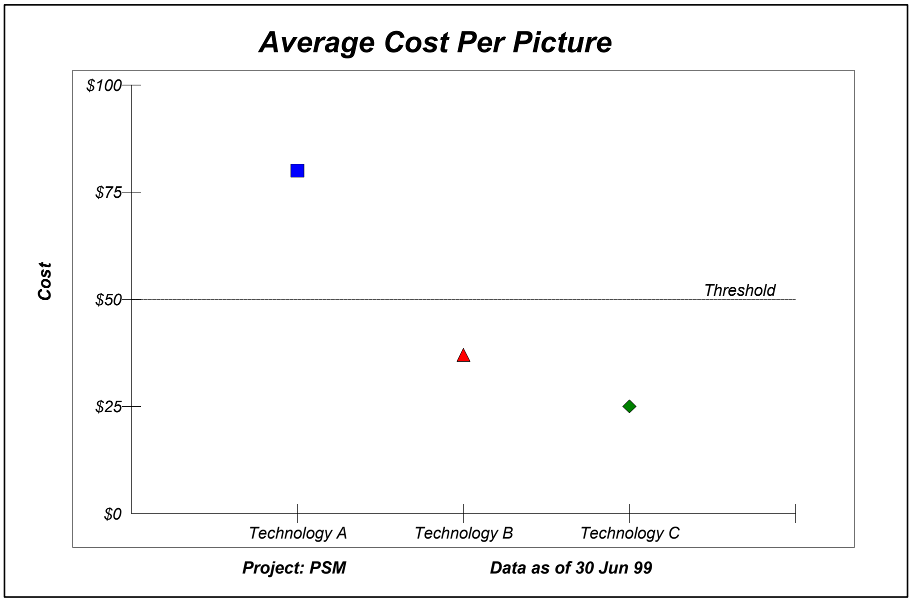

Figure 5-40b.

Finally, Figure 5-40c estimates the yearly maintenance costs of the three systems over a projected ten-year life expectancy. The costs for all systems rise over the ten-year span, but Technology B, while initially costing more in the first year, does not rise as fast as the other two. Both technologies A and C rise sharply after the third year for projected high maintenance, based upon system Mean Time Between Failure (MTBF) estimates and replacement costs.

Based on an analysis of all indicators, technology B appears to be the best choice.

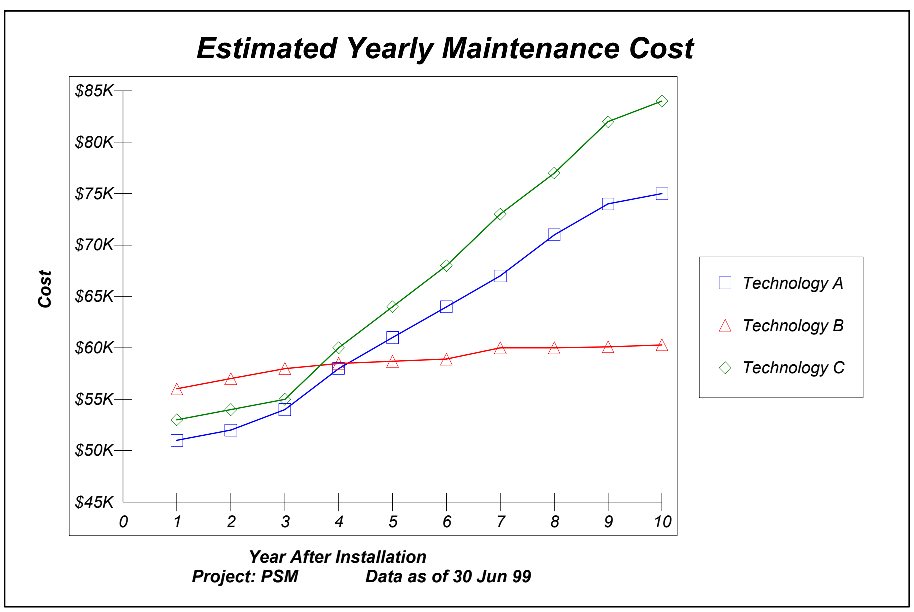

Figure 5-40c.

Additional Analysis

Combining the results for processing time, cost per picture, and maintenance cost into a single indicator, possibly using weights proportional to the importance of these factors to the customer could facilitate decision-making.

Lessons Learned

Selecting criteria appropriate to the user's needs is essential to getting a useful evaluation.

2.30 Baseline Changes

Category: Technology Volatility

Common Issue Area: Technology Effectiveness

Applicability: Applies to both software and systems engineering

Analysis Guidance and Examples

The Baseline Changes measure quantifies the number of times a specified technology changes over a period of time. Baseline Change indicators help evaluate the maturity and stability of a technology (such as an operating system or tool). This indicator can help assess the risk associated with using a technology in future development efforts.

Two examples are illustrated below. These indicators may be displayed as line charts, to show the cumulative number of releases/revisions over time, or as bar charts, to show the number of releases/revisions per year. The first example (Figures 5-41a) illustrates trends for an established technology, and the second example (Figure 5-41b) shows data from an emerging technology.

When counting baseline changes, include major baseline releases, engineering revisions with a baseline release, and patches. For example, release 1.1 equals two releases and 1.1.12 equals fourteen. This measure can be applied to vendor COTS packages and can also be used for root technologies (such as specifications) and process standards.

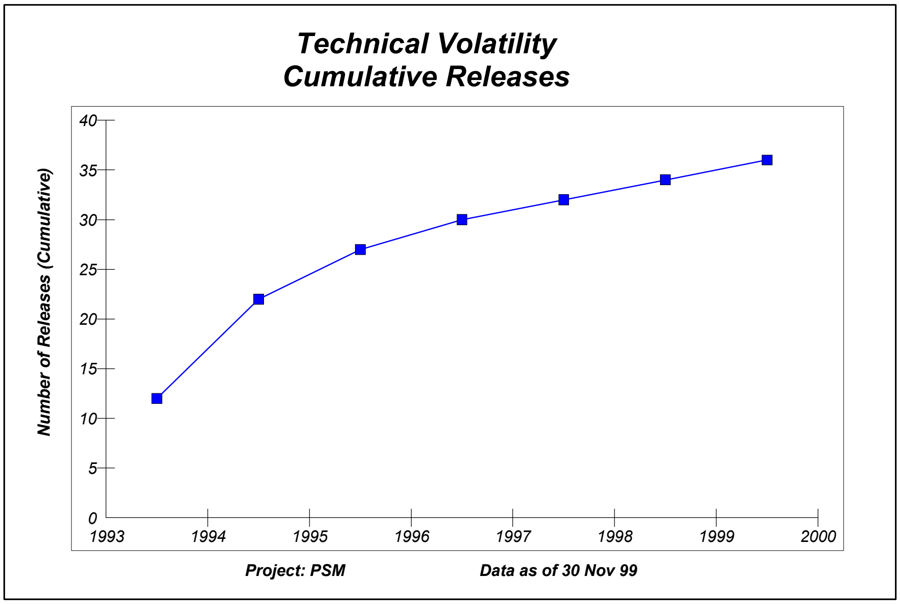

Figure 5-41a.

These indicators are particularly useful during planning, when the technology for the project is being analyzed and selected. The line graph of cumulative releases over time (Figure 5-41a) should start as a steep slope, then level out. Technology on the steep portion is changing rapidly and is immature; adopting it may be a high risk. Technology on the nearly flat portion is not changing as rapidly; it may be more mature and less risky. Similar considerations apply to the bar chart (Figure 5-41b) depicting the number of releases each year, except that the expected trend is from a high number of releases per year to a low number per year. This indicator can be used to evaluate the risk of repetitive integration problems (such as interface rework and test) as newer versions of the technology are released.

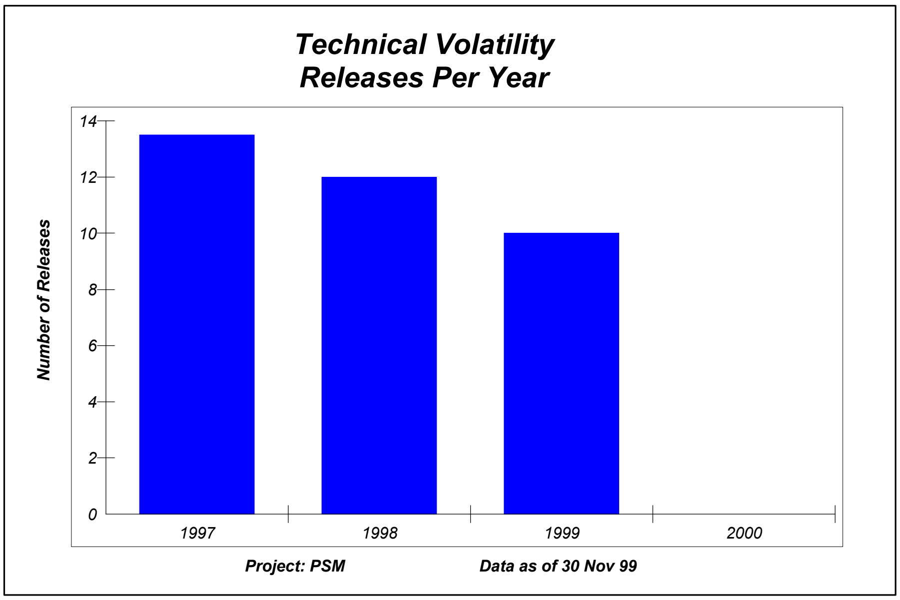

Figure 5-41b.

Additional Analysis

Data could be gathered for similar technologies under consideration. The slope and bar heights of these graphs could be compared to the other to verify the technologies' maturity or immaturity.

Lessons Learned

The newest, leading-edge technology may not have enough historical release data to produce a useful indicator.

More mature technologies have fewer releases than new technologies, and they appear at more predictable time intervals with a smaller overall impact to the project. Thus, project stability requirements may favor more mature technologies over new technologies.

2.31 Survey Results

Category: Customer Feedback

Common Issue Area: Customer Satisfaction

Applicability: Applies to most types of projects

Analysis Guidance and Examples

The Survey Results indicator represents the results of a customer satisfaction survey. Survey results are used to understand areas of customer satisfaction and dissatisfaction. When there are many customers, the survey results may be from a sample of customers. When reviewing results, consider how well that sample represents the total customer base. Survey results can be used to establish a baseline for competitive benchmarking, to identify areas in which improvements to products or services must be made, to focus marketing efforts, and to answer other specific questions.

Figure 5-42 shows a stacked bar chart of survey responses. For each question, the responses were counted in categories of Very Satisfied, Satisfied, Neutral, Dissatisfied, Very Dissatisfied, or No Response. The category totals were then converted to percent of responses. The number of survey questionnaires sent out (205) and the number returned (83) are noted in the subtitle.

Figure 5-42 shows that nearly all survey respondents were satisfied on Questions 3 and 6, but over 80 percent of the responses were Neutral, Dissatisfied, or Very Dissatisfied on Question 1. This issue must be resolved. Questions 4 and 5, with relatively few Satisfied results should also be investigated. The high proportion of Neutral responses may indicate a lack of understanding of the question, a lack of knowledge on the subject, an activity in which the customer did not participate, or a customer disinterest. Question 7 may have been confusing, because only half of those surveyed responded. Finally, Question 8 had many Very Satisfied and many Very Dissatisfied responses, implying that customers had strong opinions on the subject. A root-cause analysis may be warranted.

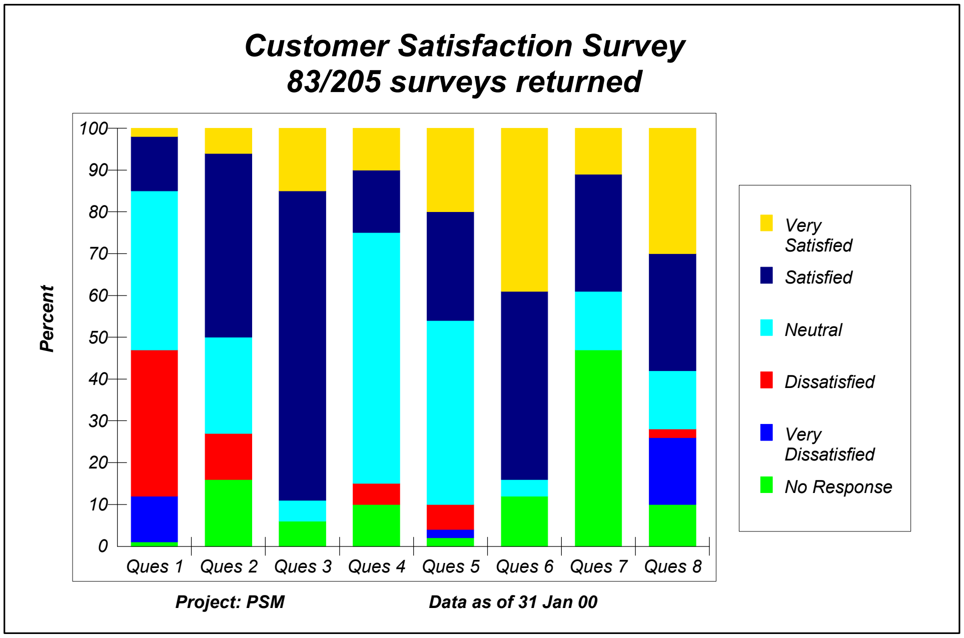

Figure 5-42.

Additional Analysis

In cases where the survey indicates dissatisfaction and there is just one customer, an interchange meeting should be held between the customer and sponsor representatives to understand the problem. With more than one customer, focus groups can sometimes provide additional data.

Used periodically throughout the project, this indicator will monitor changes in customer satisfaction. A change in survey results over time indicates the need for additional feedback to understand and address customer concerns.

If a survey is used to measure satisfaction with a prototype (such as a human-computer interface), the results may help determine if the interface is a feasible design.

Lessons Learned

Response rate can be increased by providing incentives or by using facilitated sessions to complete surveys. Survey questions should be pre-tested to ensure that questions are clear and unambiguous.

2.32 Performance Rating

Category: Customer Feedback

Common Issue Area: Customer Satisfaction

Applicability: Applies to both software and systems engineering

Analysis Guidance and Examples

The Performance Rating indicator can identify trends in the project team's performance, as perceived by customers. In its simplest form, the indicator reveals the percentage of the project's contract performance rating fee that was allocated by the customer for each evaluation period.

Performance rating indicators can be graphed in several ways. The simplest method is shown in Figure 5-43a. This graph compares the project's history of rating fees to the organization's minimum acceptable performance rating level (goal). This indicator illustrates the trend in overall customer satisfaction with the project team's performance.

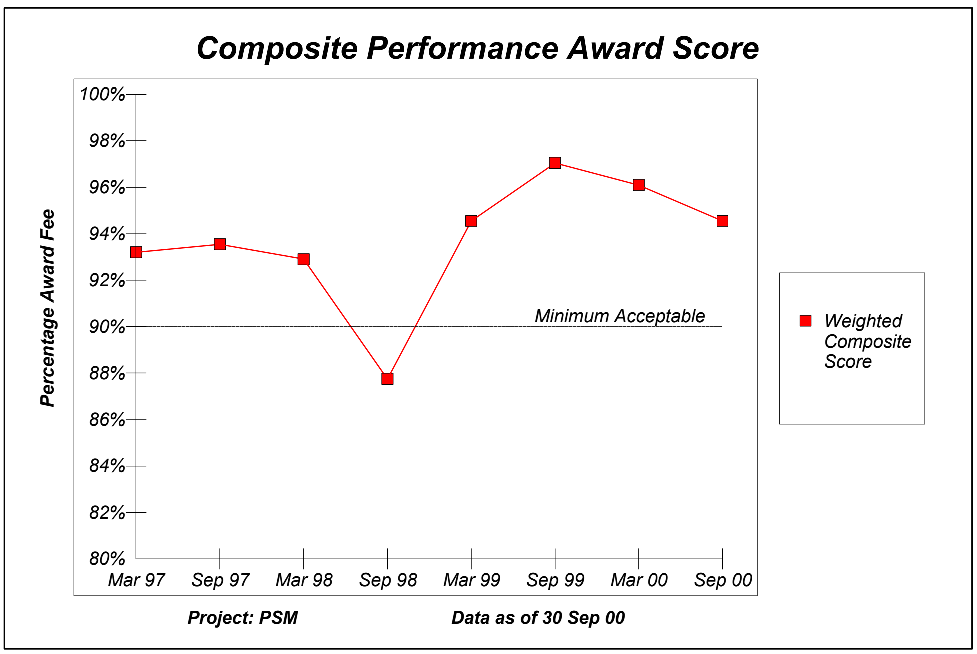

Figure 5-43a.

Figure 5-43b is an example of an indicator that shows more detail about the customer's perception of the project team's performance. It traces the history of the ratings in individual evaluation categories. A bar chart uses five data series (one for each evaluation category) within each evaluation period. In addition, the composite score (which may be a weighted sum of the individual evaluation category scores) is shown as a superimposed line graph. Analysis of this indicator reveals weak areas. In addition, trends in each evaluation category can help identify areas that need improvement or areas that are slipping.

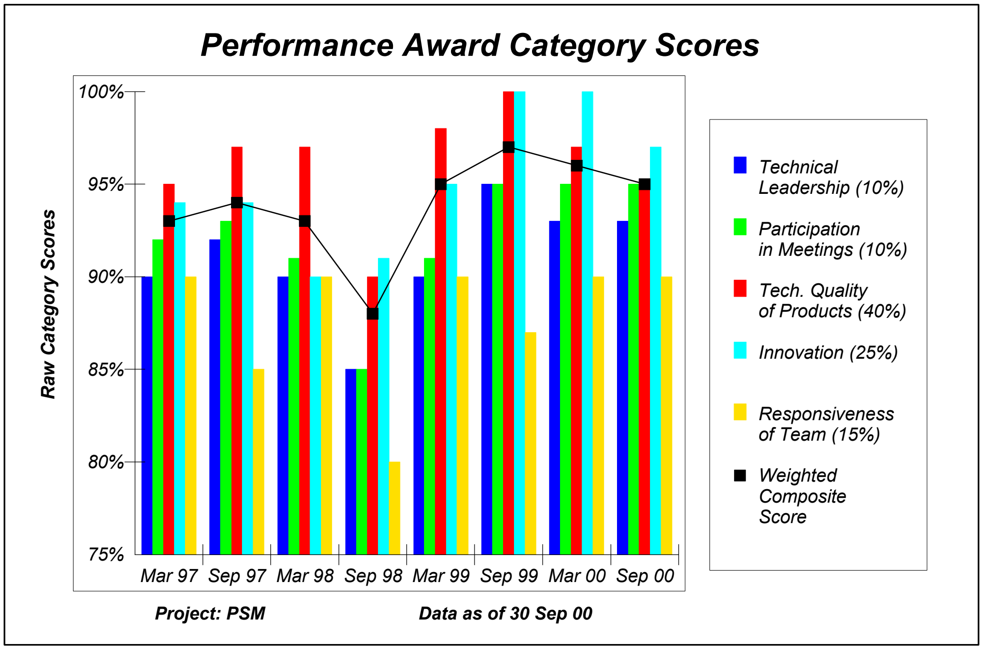

Figure 5-43b.

Additional Analysis

Comments provided with the performance-rating fee can sometimes be mapped to project or organization processes. The project or organization can use this information to identify which processes are working well and to see what improvements are needed.

Lessons Learned

The customer's method for ratings development should be consistent throughout the period illustrated by the performance rating indicators. Otherwise, the graph should be annotated to indicate where a change in the rating method occurred.

2.33 Requests for Support

Category: Customer Support

Common Issue Area: Customer Satisfaction

Applicability: Applies to most types of projects

Analysis Guidance and Examples

Requests for Support help determine how responsive the provider is to the customer and if the customer's needs and expectations are being met.

These example indicators depict data for a help desk that fields calls for technical support of a medical records system. Callers receive technical support by phone or by a technician visit. Each call is logged by time and priority (high, medium, low). If the problem is handled over the phone, a priority of "routine" is assigned. After each problem is resolved, closure time is recorded.

The stacked bar chart in Figure 5-44a shows the total number of calls received per month by priority.

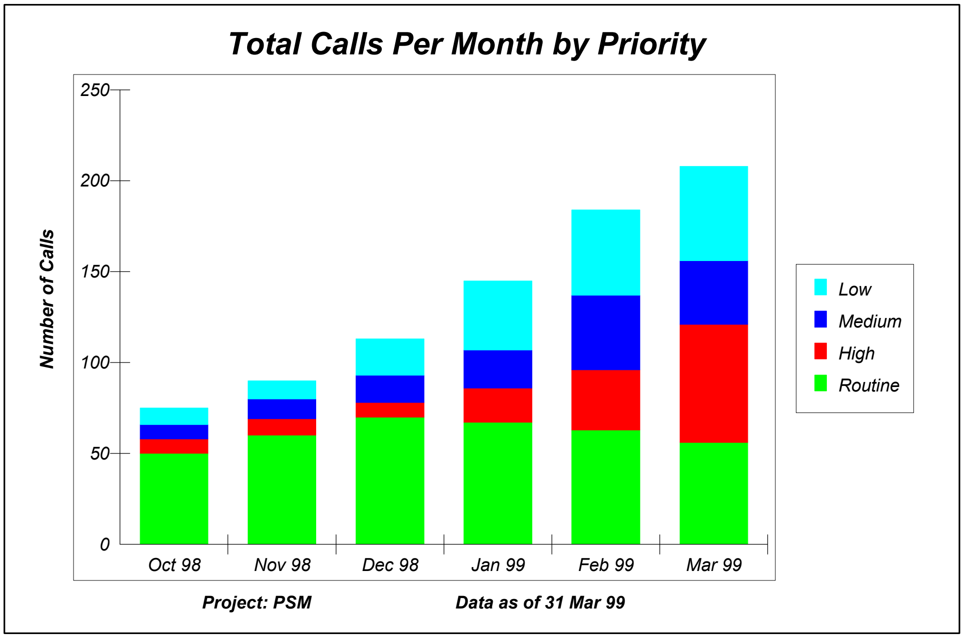

Figure 5-44a.

From this indicator, it appears that the number of trouble calls is increasing, and fewer problems are being resolved immediately by phone. This indicates that either the problems require either the personal attention of a technician or research to provide an answer. In addition, when a resolution is delayed, the number of high-priority calls increases.

Figure 5-44b shows the mean response time calculated as the length of time from when a call was received until a technician was dispatched or until a return call was made. Each priority level is graphed as a series line.

This indicator shows a trend in increasing mean response time for all priorities, with high priorities increasing faster than medium and low priorities.

The problems identified by these indicators could be due to inadequate system user training, inadequate help desk personnel training, an inadequate number of help desk personnel, declining system quality, or a combination of these.

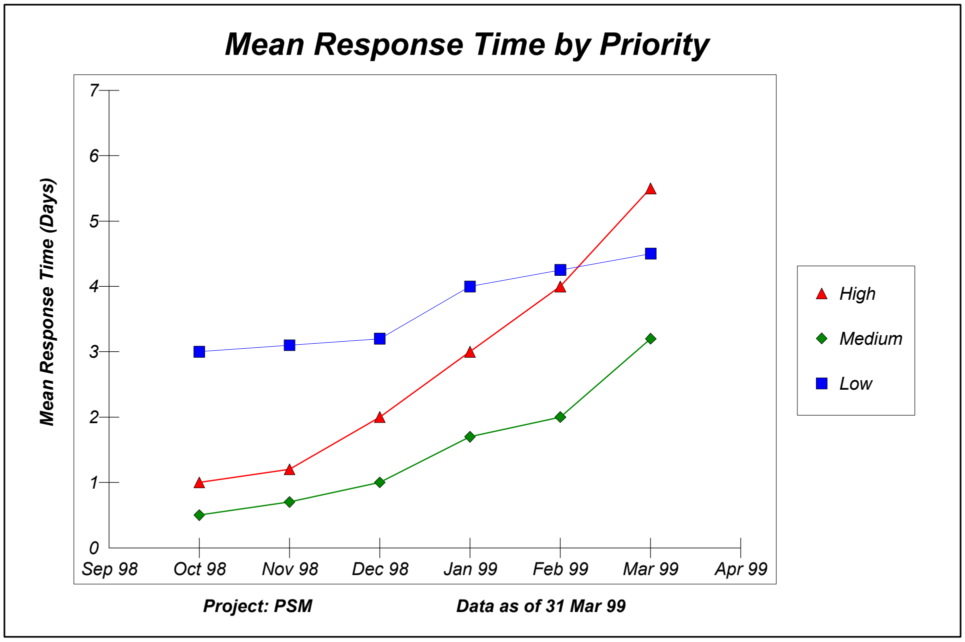

Figure 5-44b.

Additional Analysis

Each possible cause must be examined. If there is no clear cause, an analysis using more environment factors might be needed. Factors to consider include number of users, recent updates to the system, and types of calls received (hardware, software, or operator error).

Lessons Learned

Problems require careful analysis to determine if the cause is in the supported system, in the response mechanism, or in the support process. Any or all of these might be contributors.

[This page intentionally left blank.]

3

Integrated Indicator Examples

This chapter presents examples of how complex or multiple indicators are used in a thorough analysis of a particular situation or scenario. For each scenario, one or more indicators are provided, with explanations of how the indicators are analyzed together to aid in decision-making.

Note that these are examples only, and they do not represent a definitive set that should be applied to all projects.

Each scenario of multiple indicators contains the following information:

- **Measures** - the PSM measures used to generate the indicators
- **Categories** - the PSM measurement categories that best matched the information needed
- **Common Issue Area** - the PSM common issue areas that are addressed in the analysis
- **Applicability** - whether these indicators are appropriate for software and systems, or for most types of projects. Examples that apply to most types of projects include indicators for process improvement efforts
- **Analysis Guidance and Examples** - information on how the measures can be used during estimation, how the feasibility of plans can be assessed, or how project performance can be analyzed
- **Additional Analysis** - other elements to consider when using this indicator
- **Lessons Learned** - industry experience with using these indicator(s)

Table 5-45 indexes this chapter’s scenario examples. Each scenario is identified as a numbered section with its associated measures and indicators.

Scenario	***Measure(s)***	***Section***	***Indicator(s)***
Size - Effort Evaluation	• **Effort** • **Lines of Code**	**3.1**	• **Size-Effort - Estimating Relationship**
Size - Schedule Evaluation	• **Milestone Dates** • **Lines of Code**	**3.2**	• **Size-Schedule - Estimating Relationship**
Effort vs. Schedule Tradeoff	• **Milestone Dates** • **Effort**	**3.3**	• **Effort-Schedule Tradeoff**
Feasibility of Plans	• **Milestone Dates** • **Effort** • **Component Status** • **Requirements**	**3.4**	• **Planned Schedule** • **Staffing** • **Code and Unit Test Progress** • **Requirements Growth**
Design Completion	• **Component Status** • **Effort**	**3.5**	• **Design Progress by Date** • **Design Progress by Configuration Item** • **Staff Level - Plan vs. Actual** • **Staff Level by Labor Category**
Test Completion	• **Component Status** • **Defects** • **Staff Experience**	**3.6**	• **Implementation Progress** • **Test Progress - Components Successfully Tested** • **Defect Status** • **Staff Level - Test Organization**
Readiness for Delivery	• **Test Status** • **Problem Report Status** • **Failures** • **Utilization**	**3.7**	• **Test Progress** • **Problem Report Status - Open by Priority Over Time** • **System Reliability** • **CPU Utilization**
Maintenance Status	• **Requirements** • **Change Request Status** • **Failures** • **Milestone Dates**	**3.8**	• **Change Requests Implemented** • **Maintenance Requirements Stability - by Type of Change** • **System Reliability - Failure Rate** • **Milestone Progress - Maintenance Activities**
Maintainability	• **Defects** • **Cyclomatic Complexity** • **Lines of Code**	**3.9**	• **Software Quality**

Figure 5-45. Index of the Scenario Examples in this Chapter

3.1 Size - Effort Evaluation

Measures:	Effort Lines of Code
Categories:	Personnel Physical Size and Stability
Common Issue Areas:	Resources and Cost Product Size and Stability
Applicability:	Applies to software engineering

Analysis Guidance and Examples

Figure 5-46 (Putnam, et al., 1992) shows the relationship between software product size (measured in SLOC) and effort (labor hours) for one application domain. This relationship underlies many software cost models, making estimated size an important estimating and planning input.

The points on Figure 5-46 represent historical size and effort data for completed projects. Note that this relationship is exponential, but appears linear because both scales are logarithmic. This relationship is far from perfect. There is a great deal of "noise" or variance in the data that can be used to compute confidence limits. The graph shows the upper and lower 95% confidence limits, indicating that the means for 95 out of 100 projects are expected to fall within those bounds. Parametric software cost models attempt to explain some of the noise with additional input parameters that reflect project and product attributes.

The "▲" symbol represents one data point for a specific project using another means of estimating effort. Comparing a project's planning data with historical data allows an assessment of the project plan's feasibility. In the sample case, the project estimate is below the mean effort for projects of that size, but within the confidence bounds.

Graphs similar to Figure 5-46 can be used to estimate effort for a given size by using the mean value (represented by the middle line). A more conservative estimate would use the range of effort values at the upper 95% confidence limit. Using the higher estimate for effort presents less risk to the project.

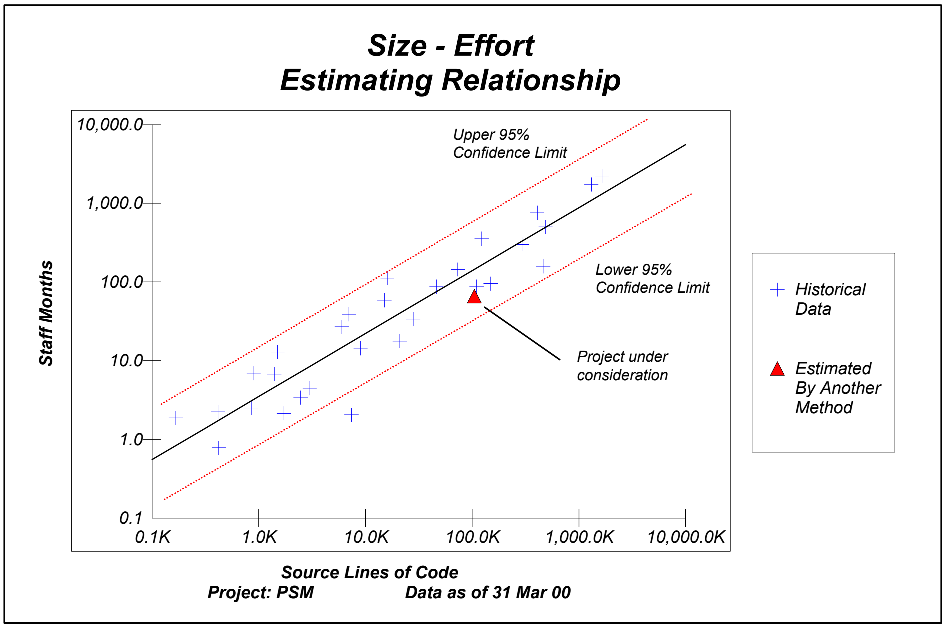

Figure 5-46.

Additional Analysis

Use this type of indicator to crosscheck effort estimates generated by other means. If estimated effort falls outside the upper or lower bound for a given application domain, the reasons for the deviation should be defined or the estimate should be adjusted.

Lessons Learned

Use caution when obtaining data for this analysis from an outside organization. Size and actual effort expended are related to staff productivity, which can vary greatly across organizations.

3.2 Size - Schedule Evaluation

Measures: Milestone Dates
Lines of Code

Categories: Milestone Performance
Physical Size and Stability

Common Issue Areas: Schedule and Progress
Product Size and Stability

Applicability: Applies to software engineering

Analysis Guidance and Examples

Figure 5-47 (Putnam, et al., 1992) shows the relationship between software product size (measured as SLOC) and schedule (duration in months) for one application domain.

The points on Figure 5-47 represent historical size and schedule data for completed projects. As in the size-effort estimating relationship (Section 3.1), the relationship between size and schedule is exponential, but appears linear because both scales are logarithmic. The graph also shows the upper and lower 95% confidence limits. The "▲" symbol represents an estimated data point for one project using another estimating method and being checked with this model.

In this figure, the current project data is above the mean effort for projects of that size, but it is well within the confidence bounds. Graphs similar to Figure 5-47 can also help generate a schedule estimate by taking the mean value for a given size, represented by the middle line. Use the range defined by the upper 95% confidence limit to be conservative.

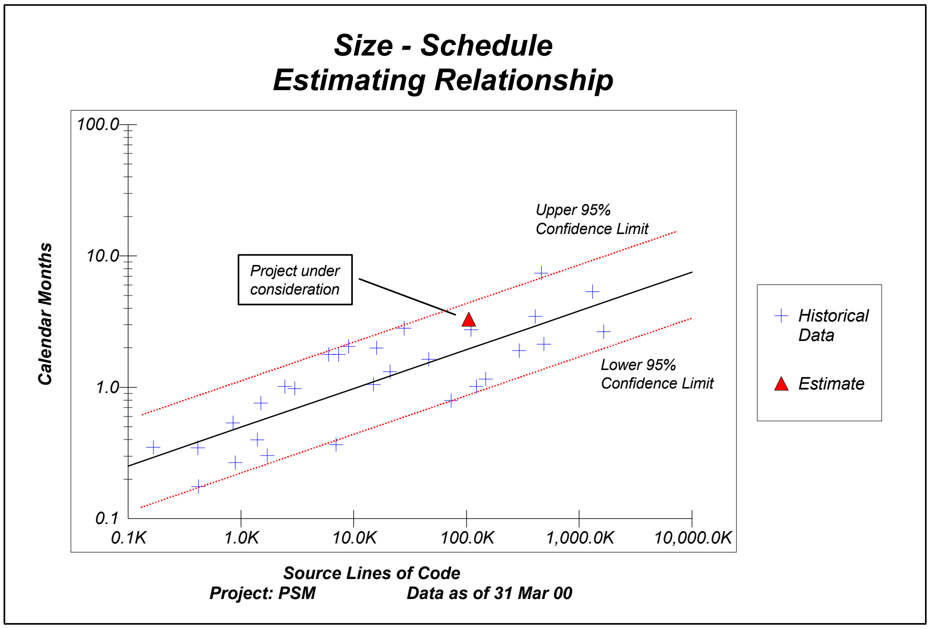

Figure 5-47.

Additional Analysis

This type of indicator can also help crosscheck estimates generated by other means. If the estimated schedule falls outside the upper or lower bound for a given application domain, the reason for the deviation should be defined or the estimate should be adjusted.

Lessons Learned

Use caution when obtaining data for this analysis from an outside organization.

3.3 Effort vs. Schedule Tradeoff

Measures:	Milestone Dates Effort
Categories:	Milestone Performance Personnel
Common Issue Areas:	Schedule and Progress Resources and Cost
Applicability:	Applies to most types of projects

Analysis Guidance and Examples

This indicator shows the tradeoff between effort and schedule, for a given project size. As the level of staffing increases, the schedule is shortened and costs increase dramatically. Figure 5-48 plots several effort-schedule combinations as the rate and peak level of staffing is varied. This indicator can help estimate the additional cost of shortening the schedule.

Figure 5-48 was adapted from Putnam and Myers, 1996. It provides a clear graphical representation of the tradeoffs between effort and schedule to develop a specific product under several plans. The schedule for each plan is plotted on the x-axis. The values on the y-axis represent the number of persons assigned to the project in each schedule month. The plotted curves illustrate each of the possible effort-schedule tradeoff plans. These curves show that costs increase in a non-linear fashion, as attempts are made to shorten the schedule by adding staff. For example, as shown in the top curve, the plan with the shortest time and highest cost to develop the system would require eight months, at a cost of $3 million. Alternatively, the plan with the longest time and lowest cost to develop the system would require 13 months, at a cost of $416 thousand. Ideally, the exact nature of an effort-schedule relationship should be determined from local historical data.

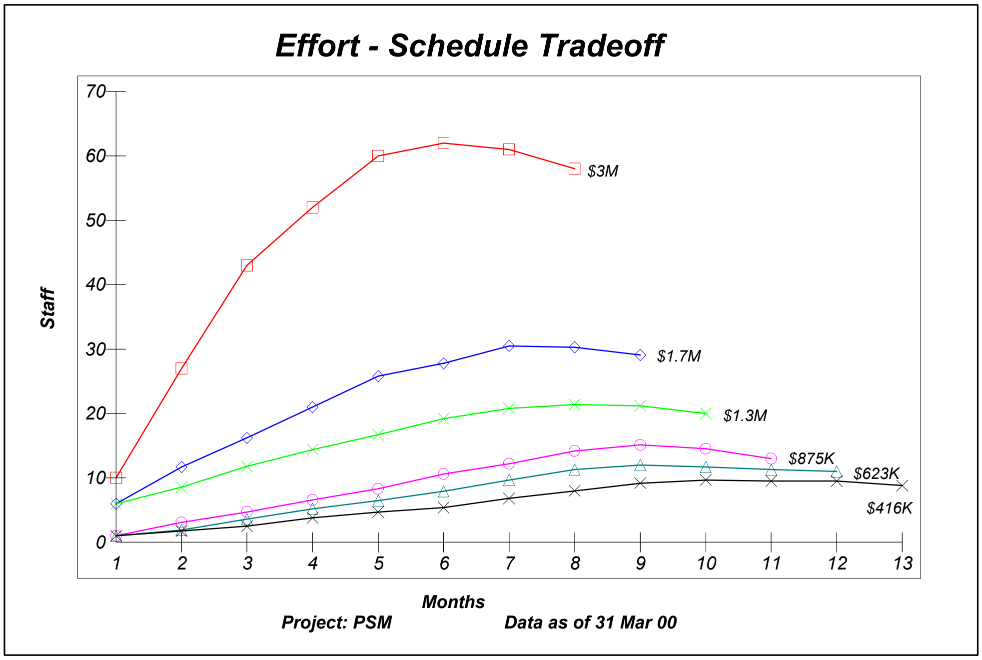

Figure 5-48.

Additional Analysis

The graph shows that a small team over an extended period of time is the appropriate strategy to minimize costs. To minimize development time, the appropriate strategy is to increase staffing, but at a dramatically increased cost.

Lessons Learned

Projects are often subjected to cost and schedule constraints established outside the project. Costs may be constrained by Congressional or corporate funding. Schedule may be set by external requirements (such as the date that a satellite is set to launch, the date that a ship will be deployed, or the date of a trade show). Effort and schedule are highly interrelated; a change in one measure will impact the other.

3.4 Feasibility of Plans

Measures: Milestone Dates
Effort
Component Status (Code and Unit Test)
Requirements

Categories: Milestone Performance
Personnel
Work Unit Progress
Functional Size and Stability

Common Issue Areas: Schedule and Progress
Resources and Cost
Product Size and Stability

Applicability: Applies to most types of projects

Analysis Guidance and Examples

Because schedule, effort, and functionality are interrelated, assumptions and plans that are associated with these attributes must be evaluated together, not just individually. For example, the question of whether or not a given schedule is feasible cannot be answered without considering the product size and the planned effort. Figures 5-49a through 5-49d represent plans from the same project that should be examined together to evaluate feasibility.

Figure 5-49a charts the planned milestone dates for major project activities.

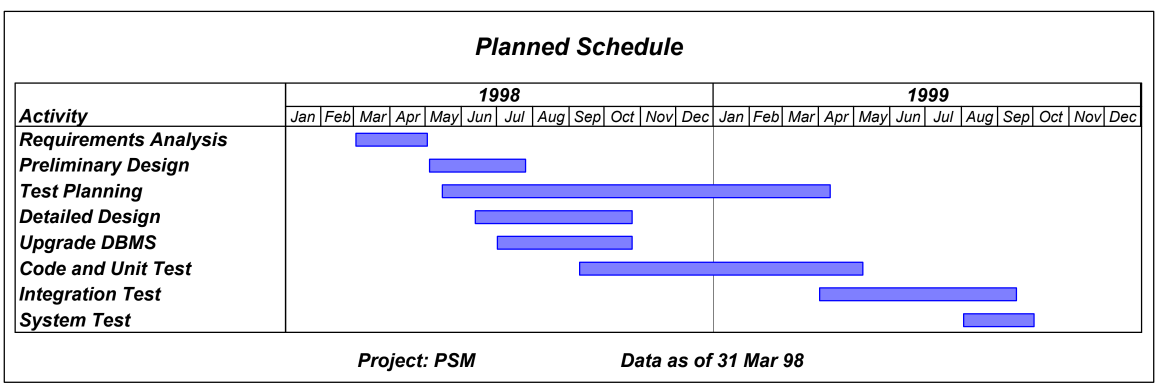

Figure 5-49a.

Figure 5-49b plots the planned staffing level over time for both the prime and the subcontractor. Note the flat staffing profile for the prime contractor. In general, a more effective staffing profile would reflect a gradual buildup during requirements analysis and the early stages of design.

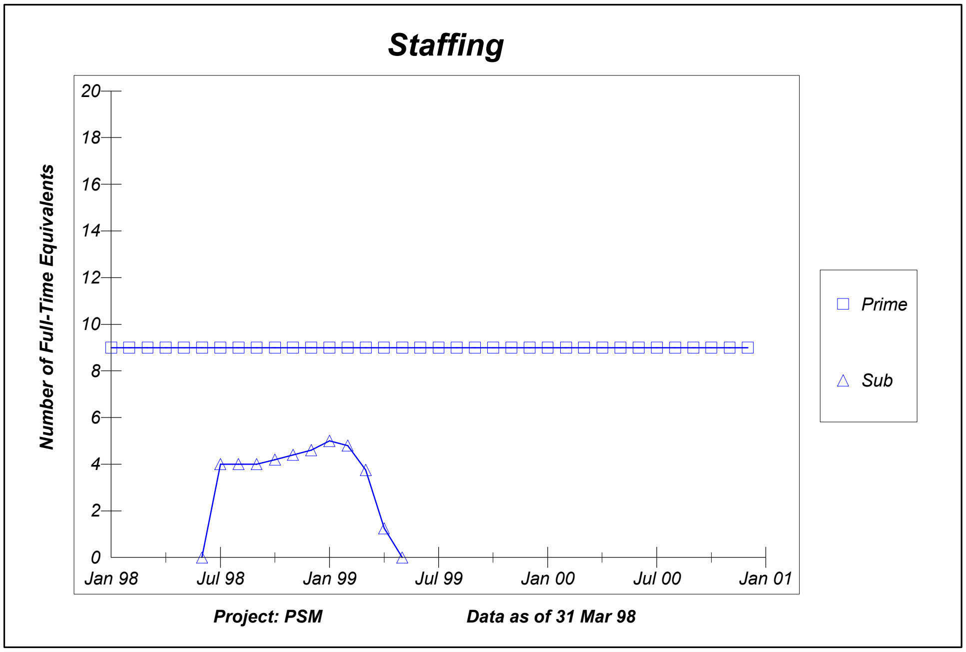

Figure 5-49b.

Figure 5-49c shows the planned work unit progress for code and unit test. Note the rapid buildup toward the end of this activity. An explanation and justification for this assumed rapid progress is needed.

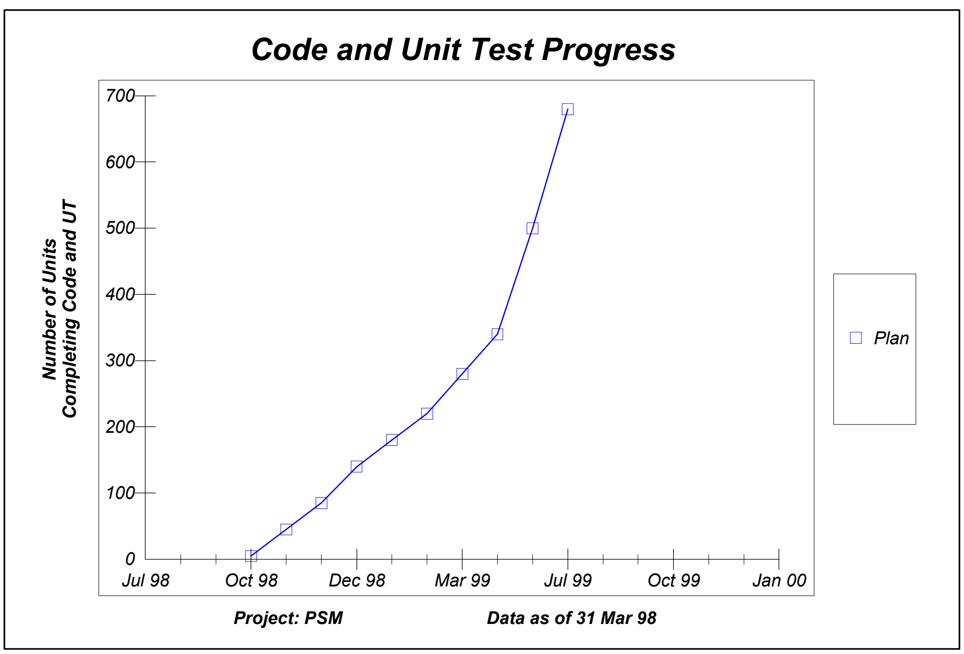

Figure 5-49c.

Figure 5-49d tracks planned growth in requirements. The assumption is that there will be a steep growth followed by absolute stability. The assumption of zero requirements growth must be questioned.

In addition to analyzing the feasibility of these four individual plans, comparing these plans yields valuable insight into overall feasibility. A review of the staffing plan identifies even more risk in the planned code and unit test progress. Note that the subcontractor provides staff only through May 1999, when code and unit test for their units is completed. The subcontractor will be unavailable to address any problems and defects found during integration and test. Note also that a rapid increase in code and unit test progress is planned after May 1999. At this time, the subcontractor is gone and the prime staffing remains constant.

By comparing the planned milestone dates with the planned requirements growth, another inconsistency is revealed. According to the milestone dates, requirement analysis will be completed by May 1998. However, the project plans assume continual requirements growth after this point.

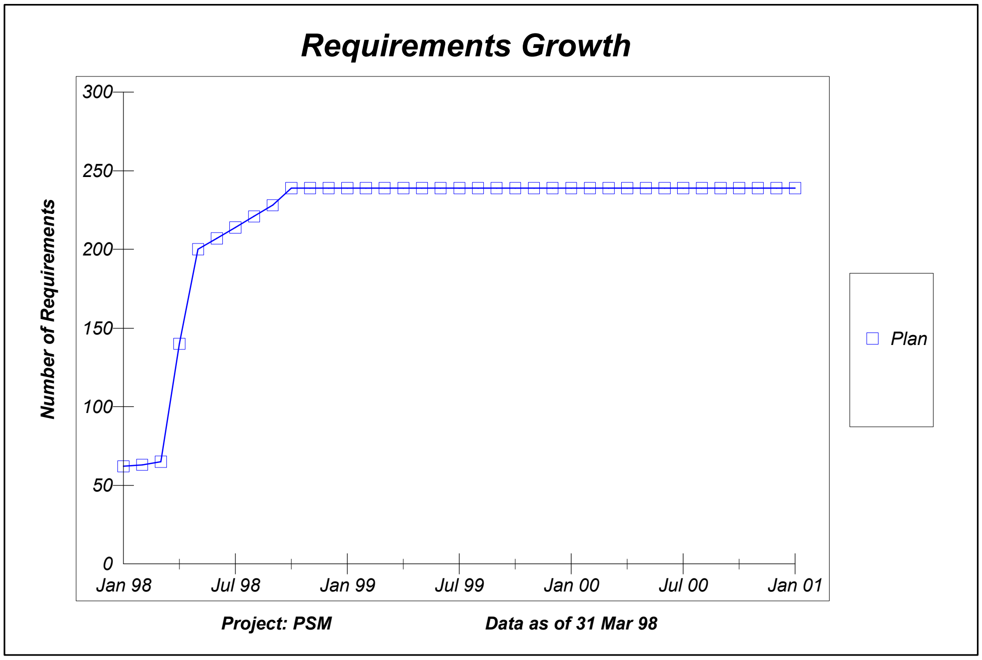

Figure 5-49d.

Additional Analysis

The next step is to refine the plans for schedule, effort, and requirements growth to develop a strategy that is realistic and internally consistent.

Lessons Learned

Once the project is underway, changes in any of the plans will likely require modification of other plans. For example, if requirements growth continues past October 1998, the other three measures must be replanned.

3.5 Design Completion

Measures:	Component Status Effort
Categories:	Work Unit Progress Personnel
Common Issue Areas:	Schedule and Progress Resources and Cost
Applicability:	Applies to most types of projects

Analysis Guidance and Examples

Projects often need to assess or predict their ability to complete the current phase in order to anticipate or uncover staffing, cost, and schedule issues associated with future phases. In this scenario, four indicators (Figures 5-50a through 5-50d) assess completion of the project's design phase; two indicators deal with the amount of work completed to date, and two indicators deal with staffing. These indicators are analyzed together to assess whether the project will complete design activities as scheduled.

The line graph in figure 5-50a compares the actual number of units completing design over time to the planned number. It indicates that actual progress is significantly behind in August. The plan is for all units to be complete by the end of September. This does not seem realistic.

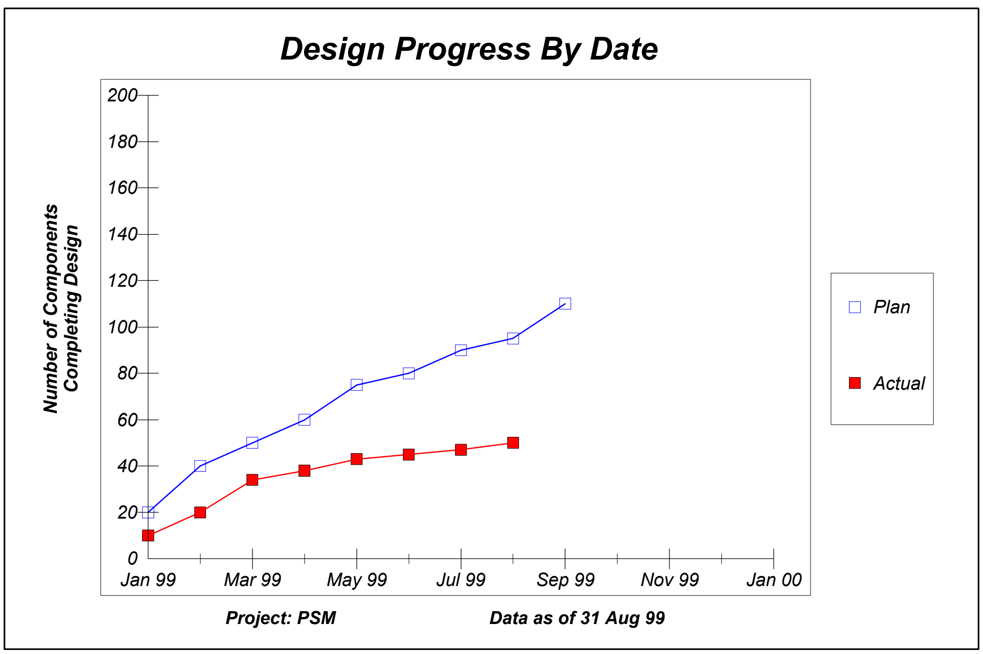

Figure 5-50a.

Figure 5-50b uses the same data as the previous figure, but the data is broken out by configuration item (CI). The data indicates that all CIs are behind schedule, but CI B is the worst.

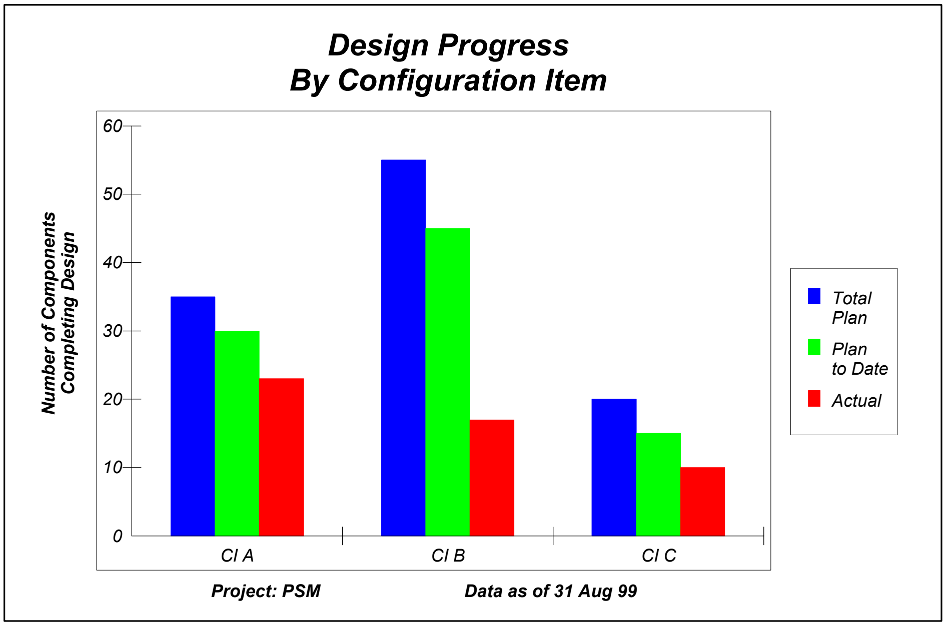

Figure 5-50b.

Figure 5-50c tracks the overall project staffing level over time. The project was significantly understaffed in May, June, and July.

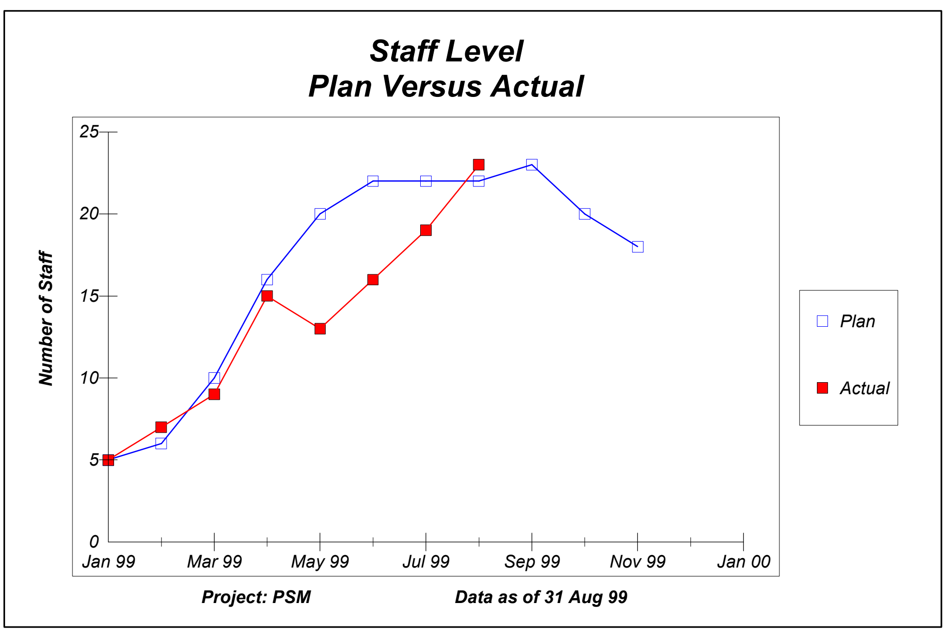

Figure 5-50c.

Figure 5-50d plots the same data by labor category. Figure 5-50c indicates that the project is currently staffed with approximately the right number of people, according to the plan. However, when actual staff is divided into labor categories, Figure 5-50d shows that the design team has fewer senior-level staff than planned.

Based on all this information, the current plan does not appear realistic. A replan for the remaining project activities is recommended, taking into consideration work remaining, current staffing levels, and current staff experience.

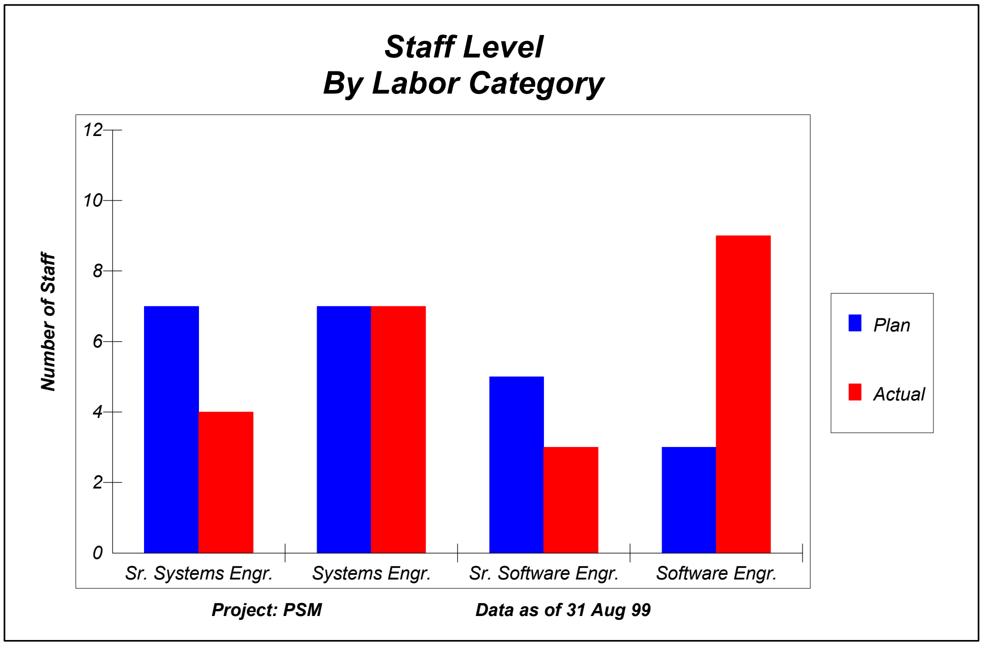

Figure 5-50d.

Additional Analysis

Additional analysis of the staff decrease in May revealed a significant turnover of experienced personnel that month. Instead of assigning new analysts to the design, the programmers scheduled to join the project in July were brought on early and assigned to the design tasks. This had a negative impact. The programmers did not have the experience to perform these tasks, and the designers had to bring the new team members up to speed.

Lessons Learned

Looking at related indicators, like staffing, helps to identify the cause of schedule problems.

3.6 Test Completion

Measures: Component Status
Defects
Staff Experience

Categories: Work Unit Progress
Functional Correctness
Personnel

Common Issue Areas: Schedule and Progress
Product Quality
Resources and Cost

Applicability: Applies to both software and systems engineering

Analysis Guidance and Examples

Two problems that often impede test completion are: 1) not receiving components to test on schedule, and 2) waiting for defects to be fixed so that components can return to test. This scenario shows how four indicators (Figures 5-51a through 5-51d) can help monitor test progress during the integration and test phase of a project.

Figure 5-51a shows that component implementation was late; consequently, component delivery to the testing group was late. While all components have been delivered, they were delivered weeks behind the original schedule.

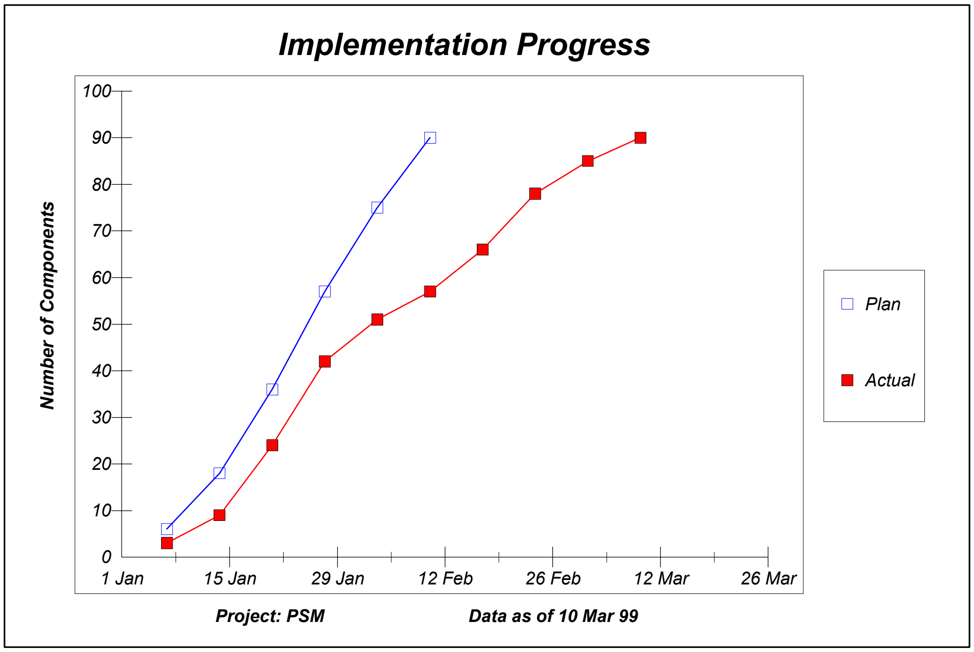

Figure 5-51a.

Figure 5-51b graphs the same components completing the next development activity, which is testing. Three progress measures are compared: 1) the original plans for component test completion, 2) components for which tests have been attempted, and 3) components that have passed testing. Figure 5-51b indicates that not as many components have been tested as originally planned, and not all of the components that were tested passed. In fact, a large number of tests failed.

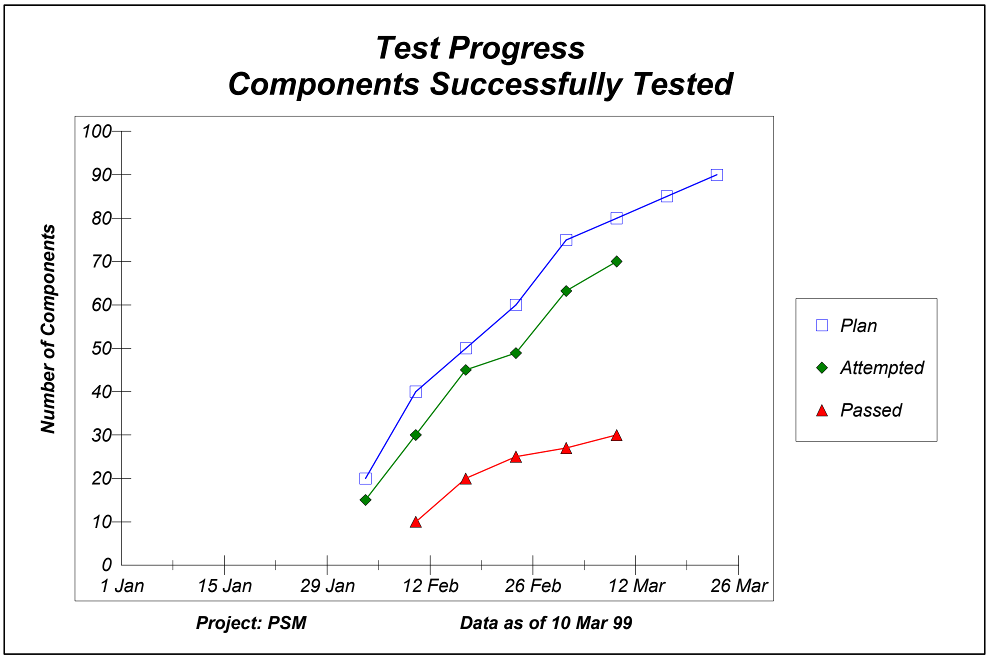

Figure 5-51b.

Figure 5-51c shows the cumulative number of defects reported and closed during the testing activity. As might be expected, testing revealed a large number of defects. The closure rate is not keeping pace with the reporting rate. Additionally, some high-priority defects are still open, which may be impacting test progress.

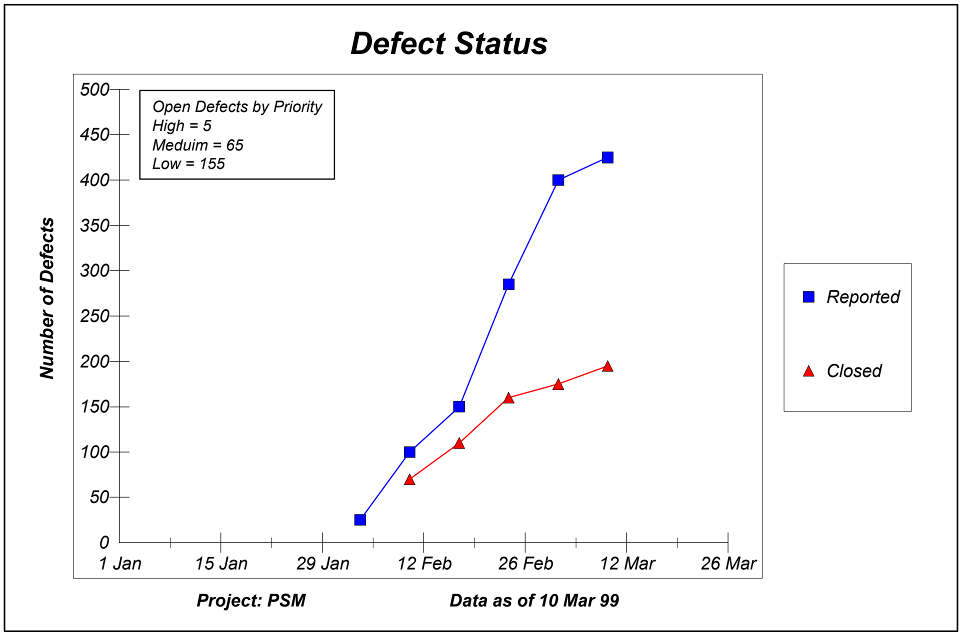

Figure 5-51c.

The final graph (Figure 5-51d) indicates that the planned reduction in testing staff was prevented by the delays. An integrated analysis of these indicators shows that test schedules and staffing plans must be revised, taking into consideration the development team's plans for fixing the outstanding defects.

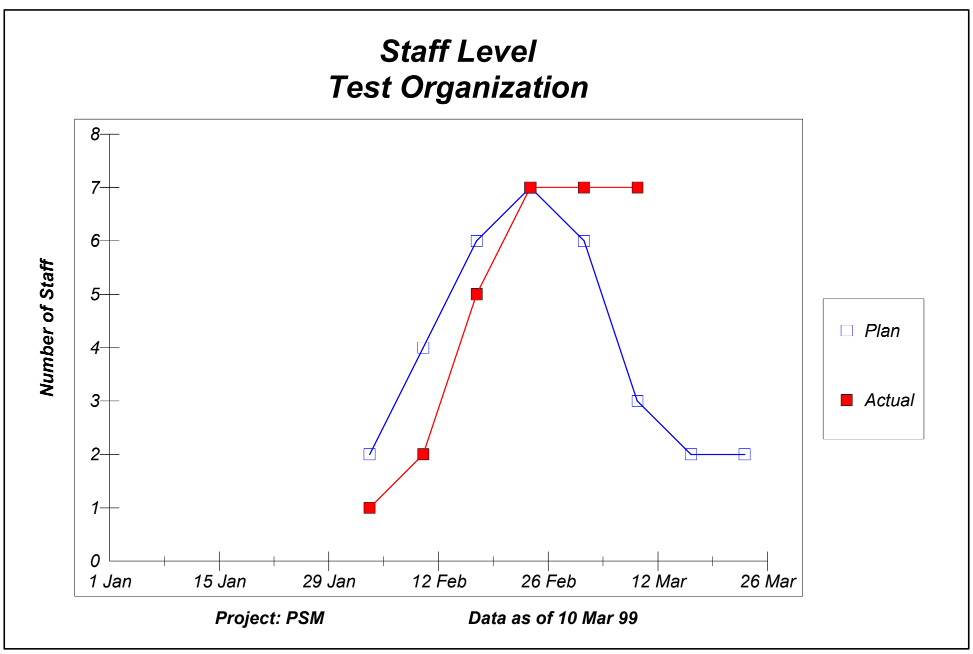

Figure 5-51d.

Additional Analysis

The analysis should also determine why the “components passed” trend line in Figure 5-51d has recently leveled off. Staff may be rushing components to test without completing inspections and other quality gates in implementation.

Lessons Learned

Development delays can have “ripple effects.” Progress problems often become leading indicators of quality problems.

3.7 Readiness for Delivery

Measures:	Test Status Problem Report Status Failures Utilization
Categories:	Work Unit Progress Dependability - Reliability Efficiency
Common Issue Areas:	Schedule and Progress Product Quality
Applicability:	Applies to both software and systems engineering

Analysis Guidance and Examples

As a system approaches its delivery date, a number of issues may influence the decision to release the product. In addition to assuring that all testing has been completed, it is often necessary to demonstrate that certain contract requirements have been met - such as meeting specified thresholds. In some cases, thresholds are implied; for example, all tests need to be run, and all high-priority problems must be fixed.

Figures 5-52a through 5-52d illustrate a set of diverse indicators that represent the specific concerns for this sample project prior to release. Figure 5-52a is a line graph of the number of successfully tested requirements. The graph reveals that testing is proceeding close to plan, with almost 80 percent of requirements tested to date.

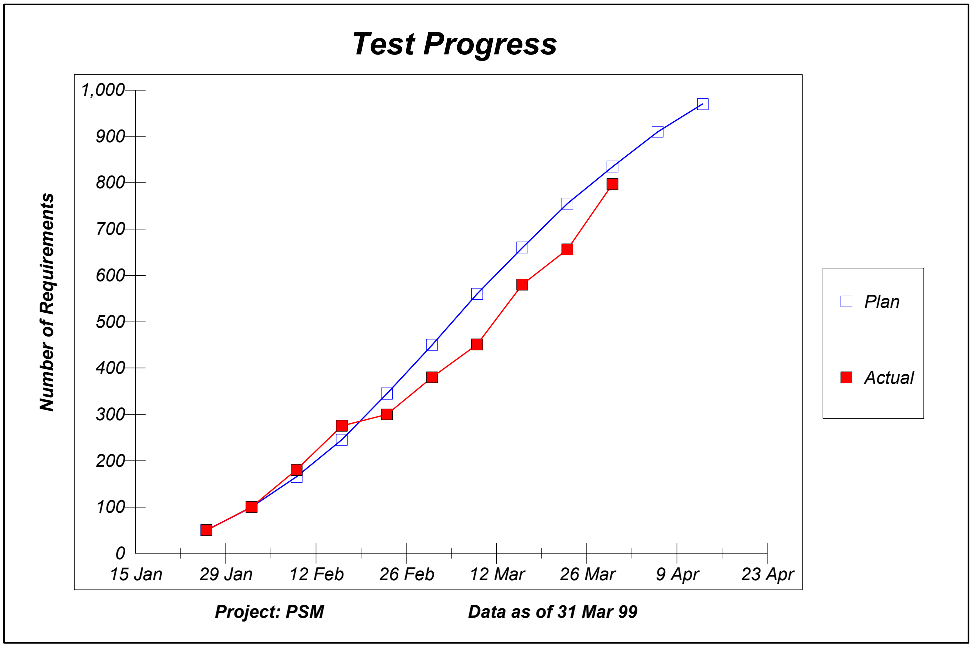

Figure 5-52a.

Figure 5-52b plots the number of open problem reports by priority. The number of open problems for all priorities is going down. Only two high-priority problems remain open. The open problems must be fixed before the system can be released.

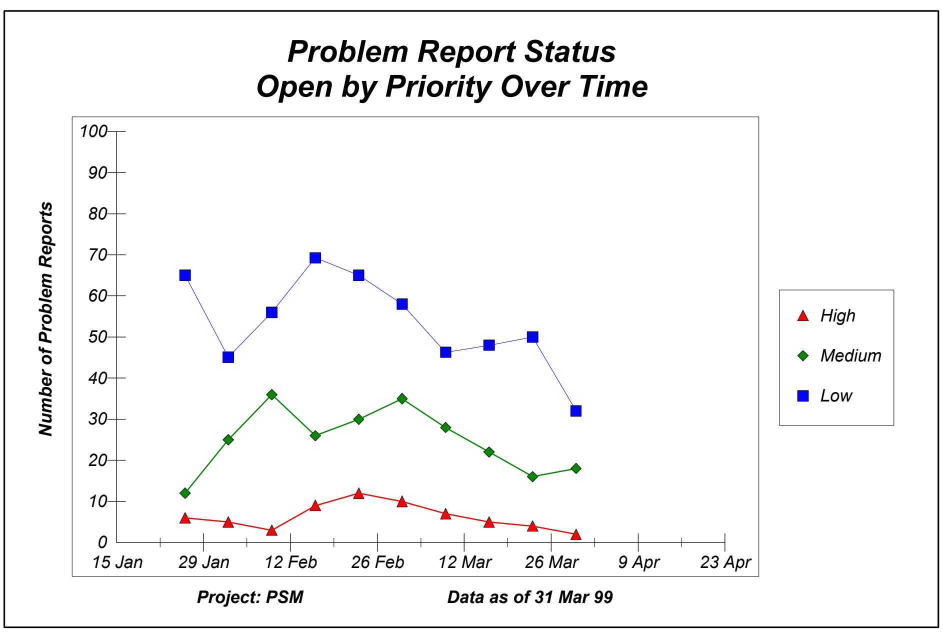

Figure 5-52b.

The line graph in Figure 5-52c shows reliability of the software during acceptance test. Reliability is measured as the mean number of hours between failures that occur each week. The time between failures is calculated by logging the total number of usage hours that elapse between failures during acceptance test. The contract requirement threshold is also indicated in the figure. This figure indicates that software reliability is approaching the acceptable minimum of 100 hours mean time between failures. The failure interval is not a cause for concern at this point.

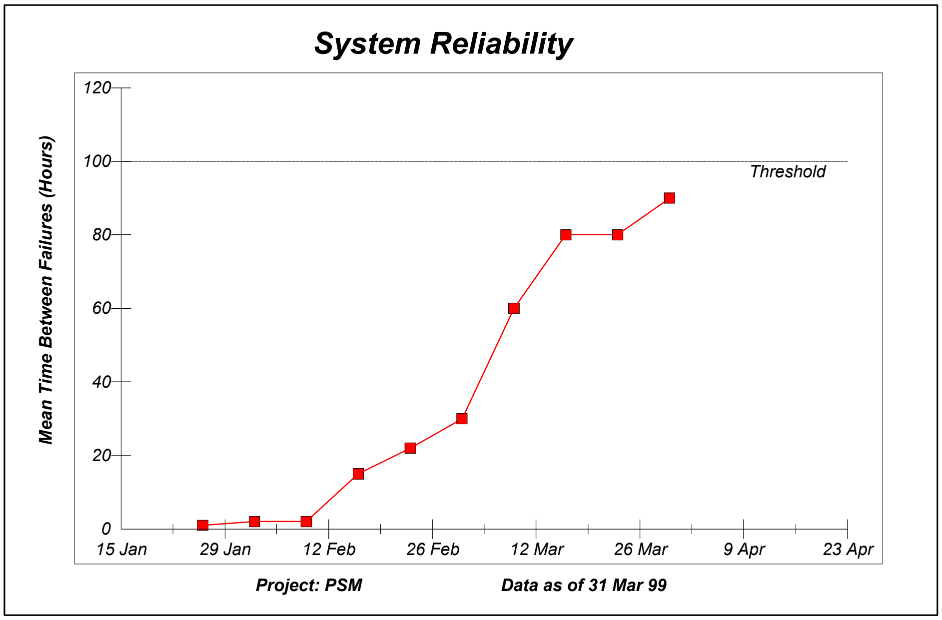

Figure 5-52c.

Figure 5-52d shows the CPU utilization of the system, measured against the contract requirement for a 50-percent reserve. This is based on a peak measurement. (Both reliability and CPU utilization are based on a user-defined operational scenario.) This figure indicates that tests show current utilization levels slightly above the 50-percent threshold.

A comparative evaluation of these four indicators reveals that the project is making steady progress in completing testing activities, that no large unplanned activities exist (as a result of rework), and that critical performance measures will probably be met. As a result, the team may proceed with plans to deliver the system as scheduled.

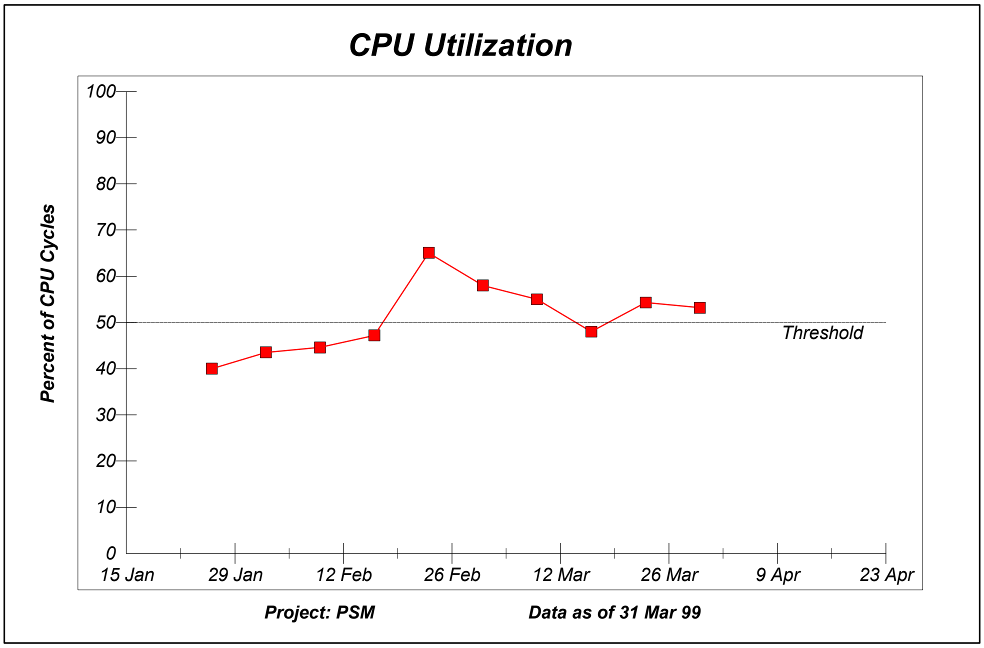

Figure 5-52d.

Additional Analysis

The remaining open problem reports should be reviewed to ensure that deferment of those problems will not adversely affect usability or key customer requirements. Any high-priority problems should be corrected prior to delivery.

Reducing the CPU utilization would probably require additional changes to some components that have otherwise been certified as working properly. This rework decision could delay delivery. The project manager and customer may decide to make a tradeoff by accepting a system that exceeds the desired threshold to allow on-time delivery. A future enhancement might address the threshold problem.

Lessons Learned

The question of what constitutes "good enough" is a difficult one. Large, complex projects will always have some defects remaining at delivery. It is essential to have quantitative criteria for the type and number of allowable defects. For example, an advanced military aircraft program should not deliver a product if any defects remain that affect safety or the ability to perform a mission.

3.8 Maintenance Status

Measures:	Requirements Change Request Status Failures Milestone Dates
Categories:	Functional Size and Stability Dependability - Reliability Milestone Performance
Common Issue Areas:	Product Size and Stability Product Quality Schedule and Progress
Applicability:	Applies to both software and systems engineering

Analysis Guidelines and Examples

System maintenance issues are often different than issues related to new software development. Figures 5-53a through 5-53d are a sample set of measurement indicators for monitoring a system that has recently entered the operations and maintenance phase. The sample system is currently on a three-month release cycle. The system has undergone three releases so far this year. Work on a fourth release is currently in progress.

Figure 5-53a is a line graph of the number of approved open change requests over time. This graph also contains bars representing the number of change requests that have been implemented (closed) in each release. When a release occurs, change requests addressed in the release are closed and removed from the backlog of approved open change requests. Between releases, new change requests increase the backlog of approved open change requests. Modifications to change requests do not impact the backlog.

Change requests can be considered maintenance requirements. As with other requirements, modifications to change requests can impact effort and schedule. Tracking the stability of maintenance requirements (change requests) helps to understand the reasons for cost and schedule variances during maintenance.

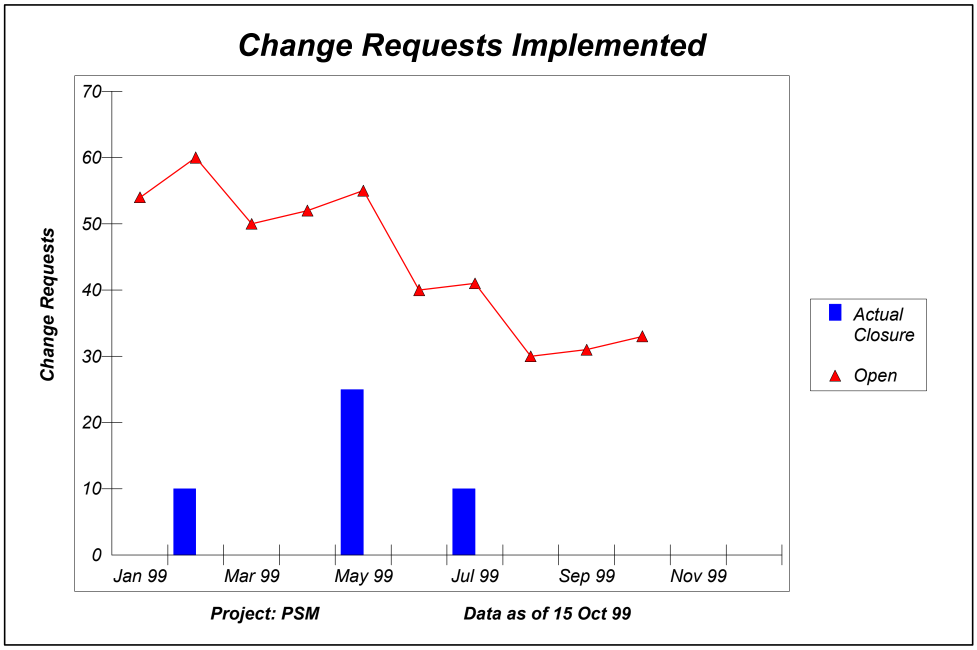

Figure 5-53a.

Figure 5-53b is a bar chart of maintenance requirements stability. Data is provided by release for requirements added, modified, or deleted. The graph indicates how the planned content of each release was affected by changing maintenance requirements prior to installation. A large number of changed requirements were associated with Release 2.

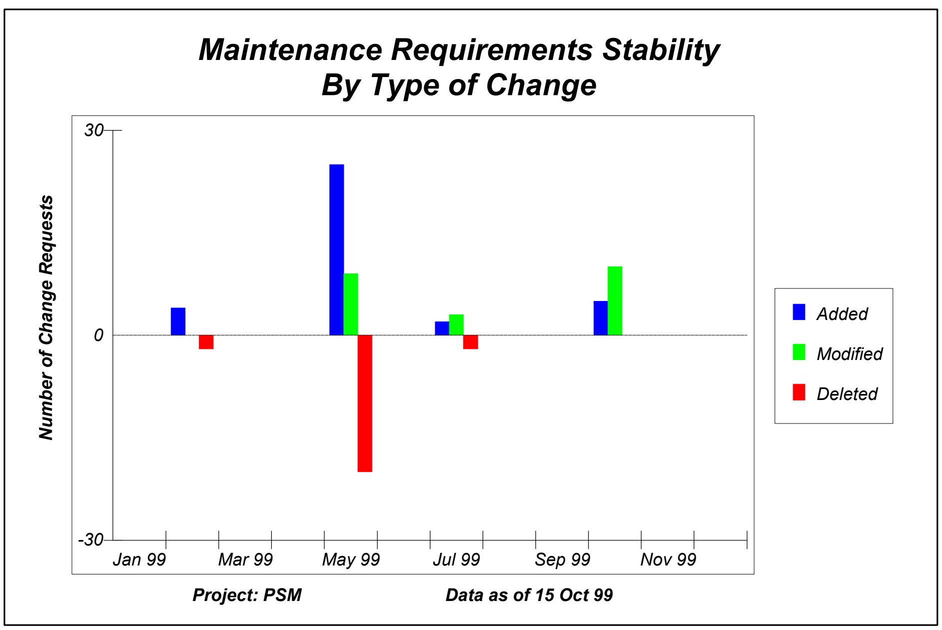

Figure 5-53b.

Figure 5-53c tracks failure rate, which is calculated by dividing the number of reported failures by the actual hours of usage between releases, normalized by 1000. The increase in failure rate in Release 2 is probably related to the volatility of its content. Releases 1 and 3 exhibit a failure rate below the desired target rate.

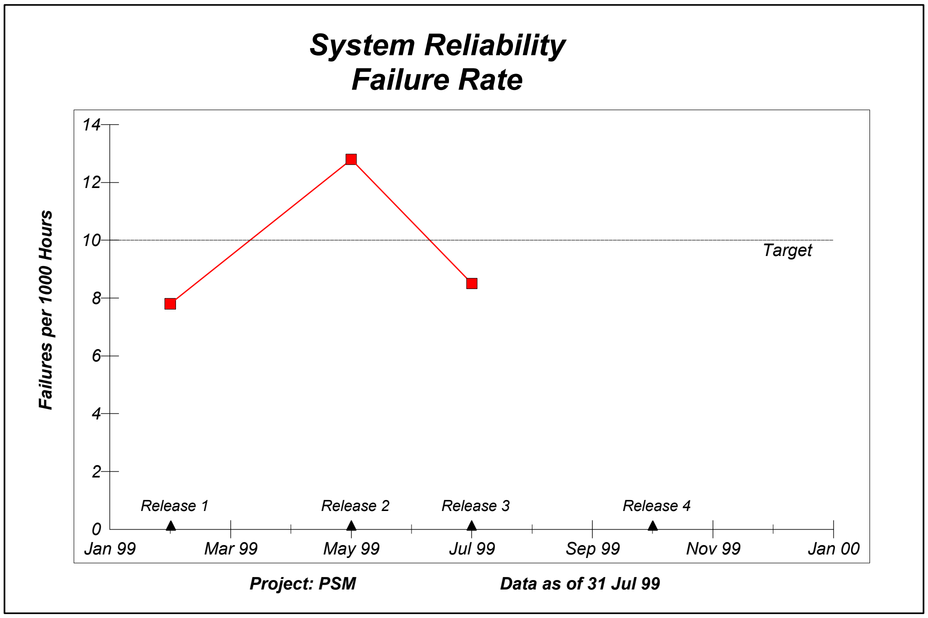

Figure 5-53c.

The final indicator (Figure 5-53d) is a milestone chart that tracks the planned and actual schedule for each release. It shows that Release 2 took longer than planned and that Release 4 is behind schedule. The delays in Releases 2 and 4 probably are due to the large number of changes in content.

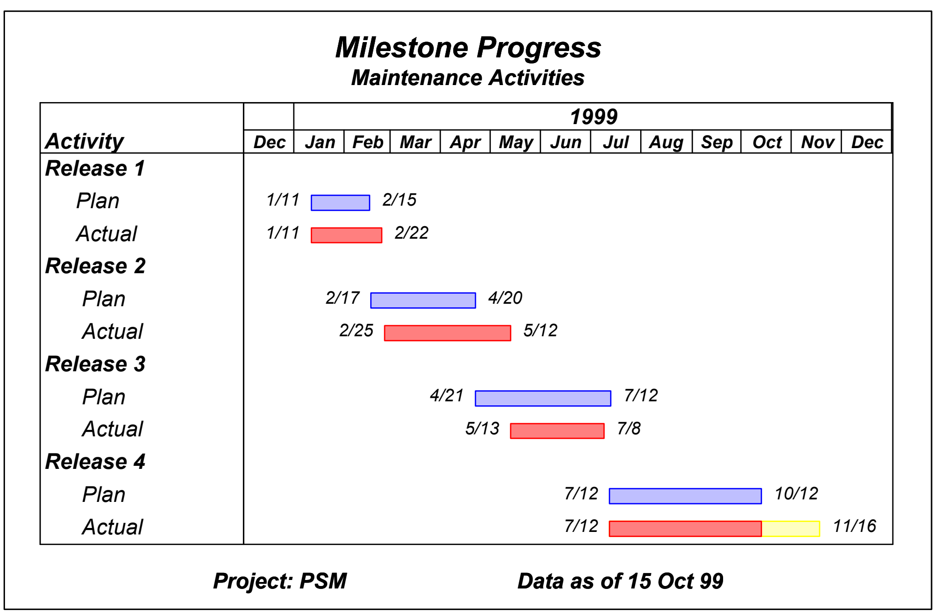

Figure 5-53d.

Additional Analysis

Since requirements volatility appears to be causing schedule delays and reliability problems, the process for release content planning should be reviewed. Tighter controls on requirements changes may be necessary.

Lessons Learned

Managing maintenance requires indicators similar to, but different from those used in development.

3.9 Maintainability

Measures:	Defects Cyclomatic Complexity Lines of Code
Categories:	Functional Correctness Supportability - Maintainability Physical Size and Stability
Common Issue Areas:	Product Quality Product Size and Stability
Applicability:	Applies to software engineering

Analysis Guidance and Examples

Once a system is developed, software maintainability problems can be identified by investigating software quality factors that are known to influence maintainability (such as size, code complexity, and defect density). If maintainability and maintenance costs are an issue, system components should be evaluated to identify any components that need additional attention.

Figure 5-54 is used to analyze an existing system, by reporting quality measures in tabular form by CI. This table identifies which CIs have the most problems related to quality, and therefore, should be the hardest to maintain

The Navigation CI has the highest defect density, the highest average complexity, and the largest number of units with a complexity of more than 10. System Services also has high numbers across all measures. These CI may be expected to require substantial maintenance support.

While Display Services has a relatively high average complexity, it does not have many defects relative to size. While some of the units are complex, they are relatively straightforward case statements, and, are not of high concern.

Software Quality

CI	Size (KSLOC)	Total Valid Defects	Defect Density	Number of Units	Average Complexity	Units with Complexity >10
Navagation	124.8	42	0.34	156	9.2	26
Sonar	45.6	12	0.26	68	6.7	15
Weapons	56.6	14	0.25	75	5.2	8
System Services	75.3	20	0.27	102	7.5	16
Display Services	168.0	32	0.19	125	8.6	12
Training	25.5	3	0.12	42	4.2	3
Total / Average	495.8	123	0.25	568	6.9	80

Project: PSM *Data as of 31 March 98*

Figure 5-54.

Additional Analysis

Defect densities can be generated at lower structure levels to identify specific components that require more quality control or redesign. The overall quality of a development project can often be evaluated by looking at the first six to twelve months of post-release defect densities. Large numbers of field-reported defects may be the result of unachieved requirements, inadequate tests, or poor code quality.

Lessons Learned

Many factors affect maintainability. Focus on the most significant.

[This page intentionally left blank.]